Competitive Programming Library

@iagorrr

April 18, 2025

Contents

1 Algorithms

1.1 Count inversions

Description: Count the number of inversions when transforming the vector l in the vector r, which is also equivalent to the mininum number of swaps required.
Usage: If no r vector is provided it considers r as the sorted vector, if there is no such way to turn l into r using swaps then -1 is returned
Time: $O(N \log N)$

```
#pragma once
#include "../Contest/template.cpp"
template <typename T>
ll countInversions(vector<T> l, vector<T> r = {}) {
    if (!len(r)) r = l, sort(all(r));
    int n = len(l);
    vi v(n), bit(n);
    vector<pair<T, int>> w;
    rep(i, 0, n) w.eb(r[i], i + 1);
    sort(all(w));
    rep(i, 0, n) {
        auto it = lower_bound(all(w), make_pair(l[i], 0));
        if (it == w.end() or it->first != l[i]) return -1;  // impossible
        v[i] = it->second;
        it->second = -1;
    }
    ll ans = 0;
    rrep(i, n - 1, 0 - 1) {
        for (int j = v[i] - 1; j; j -= j & -j) ans += bit[j];
        for (int j = v[i]; j < n; j += j & -j) bit[j]++;
    }
    return ans;
}
```

1.2 Ternary search (integer)

Description: Given a unimodal function f defined between the integers l and r finds an x such that f(x) is maximum/minimum.
Usage: Just pass the range l, r of the function you are interested, the function that receives an integer and if you want the maximum value use the cmp = greater<ll>(), otherwise less<ll>().
Time: $O(\log r - l + 1)$
Memory: $O(1)$

```
#include "../Contest/template.cpp"
template <auto cmp = greater<ll>()>
ll ternary_search(ll l, ll r, function<ll(ll)> f) {
    static const ll eps = 3;
    while (r - l >= eps) {
        ll m1 = l + (r - l) / 3;
        ll m2 = r - (r - l) / 3;
        if (cmp(f(m1), f(m2)))
            r = m2;
        else
            l = m1;
    }
    rep(i, l, r + 1) if (cmp(f(i), f(l))) l = i;
    return l;
}
```

1.3 Ternary search (real)

```
#include "../Contest/template.cpp"
template <auto cmp = greater<ld>()>
ld ternarySearch(ld l, ld r, function<ld(ld)> f, const ld eps = 1e-9) {
    while (r - l >= eps) {
        ld m1 = l + (r - l) / 3;
        ld m2 = r - (r - l) / 3;
        if (cmp(f(m1), f(m2)))
            r = m2;
        else
            l = m1;
    }
    return l;
}
```

2 Combinatorics

2.1 Process all partitions of a set

Description: generate every distinct group of a set that contains elements from 0 to N-1, and pass it to the given function "process". If N is 4 the sets generated would be :
[{{0,1,2,3}}] [{{0,1,2},{3}}] [{{0,1,3},{2}}] [{{0,1},{2,3}}] [{{0,1},{2},{3}}] [{{0,2,3},{1}}] [{{0,2},{1,3}}] [{{0,2},{1},{3}}] [{{0,3},{1,2}}] [{{0},{1,2,3}}] [{{0},{1,2},{3}}] [{{0,3},{1},{2}}] [{{0},{1,3},{2}}] [{{0},{1},{2,3}}] [{{0},{1},{2},{3}}]
Time: $O(B(N))$, Bell Number of N
Memory: $O(N)$

```
#include "../Contest/template.cpp"
void process_all_partitions_of_a_set(
    const int N, const function<void(const vi2d &)> process) {
    vi2d groups;
    groups.reserve(N);
    function<void(int)> _dfs = [&](int idx) {
        if (idx == N) {
            process(groups);
            return;
        }
        rep(i, 0, len(groups)) {
            groups[i].eb(idx);
            _dfs(idx + 1);
            groups[i].ppb();
        }
        groups.pb({idx});
        _dfs(idx + 1);
```

```
        groups.ppb();
    };
    _dfs(0);
}
```

3 Contest

3.1 bash config

```
#copy first argument to clipborad ! ONLY WORK ON XORG !
alias clip="xclip -sel clip"
# compile the $1 parameter, if a $2 is provided
# the name will be the the binary output, if
# none is provided the binary name will be
# 'a.out'
comp() {
  echo ">> COMPILING $1 <<" 1>&2
  if [ $# -gt 1 ]; then
    outfile="${2}"
  else
    outfile="a.out"
  fi
  time g++ -std=c++20 \
    -O2 \
    -g3 \
    -Wall \
    -fsanitize=address,undefined \
    -fno-sanitize-recover \
    -D LOCAL \
    -o "${outfile}" \
    "$1"
  if [ $? -ne 0 ]; then
    echo ">> FAILED <<" 1>&2
    return 1
  fi
  echo ">> DONE << " 1>&2
}
# run the binary given in $1, if none is
# given it will try to run the 'a.out'
# binary
run() {
  to_run=./a.out
  if [ -n "$1" ]; then
    to_run="$1"
  fi
  time $to_run
}
# just comp and run your cpp file
# accpets <in1 >out and everything else
comprun() {
  comp "$1" "a" && run ./a ${@:2}
}
testall() {
  comp "$1" generator
  comp "$2" brute
  comp "$3" main
  input_counter=1
  while true; do
    echo "$input_counter"
    run ./generator >input
    run ./main <input >main_output.txt
    run ./brute <input >brute_output.txt
    diff brute_output.txt main_output.txt
    if [ $? -ne 0 ]; then
      echo "Outputs differ at input $input_counter"
      echo "Brute file output:"
      cat brute_output.txt
      echo "Main file output:"
      cat main_output.txt
      echo "input used: "
      cat input
      break
    fi
    ((input_counter++))
  done
}
touch_macro() {
  cat "$1"/template.cpp >"$2"
  cat "$1"/run.cpp >>"$2"
  cp "$1"/debug.cpp .
}
# Creates a contest with hame $2
# Copies the macro and debug file from $1
# Already creates files a...z .cpp and .py
prepare_contest() {
  mkdir "$2"
  cd "$2"
  for i in {a..z}; do
    touch_macro $1 $i.cpp
  done
}
get_file_hash() {
  local hash=$(cpp -dD -P -fpreprocessed "$1" | tr -d '[:space:]' | md5sum
    | cut -c-6)
  echo "$hash"
}
```

3.2 debug

```
template <typename T>
concept Printable = requires(T t) {
    { std::cout << t } -> std::same_as<std::ostream &>;
};
template <Printable T>
void __print(const T &x) {
```

```cpp
    cerr << x;
}
template <size_t T>
void __print(const bitset<T> &x) {
    cerr << x;
}
template <typename A, typename B>
void __print(const pair<A, B> &p);
template <typename... A>
void __print(const tuple<A...> &t);
template <typename T>
void __print(stack<T> s);
template <typename T>
void __print(queue<T> q);
template <typename T, typename... U>
void __print(priority_queue<T, U...> q);
template <typename A>
void __print(const A &x) {
    bool first = true;
    cerr << '{';
    for (const auto &i : x) {
        cerr << (first ? "" : ","), __print(i);
        first = false;
    }
    cerr << '}';
}
template <typename A, typename B>
void __print(const pair<A, B> &p) {
    cerr << '(';
    __print(p.first);
    cerr << ',';
    __print(p.second);
    cerr << ')';
}
template <typename... A>
void __print(const tuple<A...> &t) {
    bool first = true;
    cerr << '(';
    apply(
        [&first](const auto &...args) {
            ((cerr << (first ? "" : ","), __print(args), first = false),
    ...);
        },
        t);
    cerr << ')';
}
template <typename T>
void __print(stack<T> s) {
    vector<T> debugVector;
    while (!s.empty()) {
        T t = s.top();
        debugVector.push_back(t);
        s.pop();
    }
    reverse(debugVector.begin(), debugVector.end());
    __print(debugVector);
}
template <typename T>
void __print(queue<T> q) {
    vector<T> debugVector;
    while (!q.empty()) {
        T t = q.front();
        debugVector.push_back(t);
        q.pop();
    }
    __print(debugVector);
}
template <typename T, typename... U>
void __print(priority_queue<T, U...> q) {
    vector<T> debugVector;
    while (!q.empty()) {
        T t = q.top();
        debugVector.push_back(t);
        q.pop();
    }
    __print(debugVector);
}
void _print() { cerr << "]\n"; }
template <typename Head, typename... Tail>
void _print(const Head &H, const Tail &...T) {
    __print(H);
    if (sizeof...(T)) cerr << ", ";
    _print(T...);
}
#define dbg(x...)                  \
    cerr << "[" << #x << "] = ["; \
    _print(x)
```

3.3 run

```cpp
void run();
int32_t main() {
#ifndef LOCAL
    fastio;
#endif
    int T = 1;
    cin >> T;
    rep(t, 0, T) {
        dbg(t);
        run();
    }
}
void run() {}
```

3.4 short-template

```cpp
#include <bits/stdc++.h>
```

```cpp
using namespace std;
#define fastio                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
void run() {}
int32_t main(void) {
    fastio;
    int t;
    t = 1;
    // cin >> t;
    while (t--) run();
}
```

3.5 template

```cpp
#pragma once
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif
#define fastio                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
#define all(j) j.begin(), j.end()
#define rall(j) j.rbegin(), j.rend()
#define len(j) (int)j.size()
#define rep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i < (b); i++)
#define rrep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i > (b); i--)
#define trav(xi, xs) for (auto &xi : xs)
#define rtrav(xi, xs) for (auto &xi : ranges::views::reverse(xs))
using ll = long long;
#define endl '\n'
#define pb push_back
#define pf push_front
#define ppb pop_back
#define ppf pop_front
#define eb emplace_back
#define ef emplace_back
#define lb lower_bound
#define ub upper_bound
#define fi first
#define se second
#define emp emplace
#define ins insert
#define divc(a, b) ((a) + (b) - 1ll) / (b)
using str = string;
using ull = unsigned long long;
using ld = long double;
using vll = vector<ll>;
using pll = pair<ll, ll>;
using vll2d = vector<vll>;
using vi = vector<int>;
using vi2d = vector<vi>;
using pii = pair<int, int>;
using vpii = vector<pii>;
using vc = vector<char>;
using vs = vector<str>;
template <typename T, typename T2>
using umap = unordered_map<T, T2>;
template <typename T>
using pqmn = priority_queue<T, vector<T>, greater<T>>;
template <typename T>
using pqmx = priority_queue<T, vector<T>>;
template <typename T, typename U>
inline bool chmax(T &a, U const &b) {
    return (a < b ? a = b, 1 : 0);
}
template <typename T, typename U>
inline bool chmin(T &a, U const &b) {
    return (a > b ? a = b, 1 : 0);
}
template <typename T>
std::istream &operator>>(std::istream &is, std::vector<T> &vec) {
    for (auto &element : vec) {
        is >> element;
    }
    return is;
}
template <typename T>  // print vector
ostream &operator<<(ostream &os, vector<T> &xs) {
    rep(i, os.iword(0), xs.size()) os << xs[i] << (i == xs.size() ? "" : "
     ");
    os.iword(0) = 0;
    return os;
}
```

3.6 vim config

```vim
set sta nu rnu sc cindent noswapfile
set ts=2 sw=2
set bg=dark ruler clipboard=unnamed,unnamedplus, timeoutlen=100
colorscheme default
syntax on
" Takes the hash of the selected text and put
" in the vim clipboard
function! HashSelectedText()
    " Yank the selected text to the unnamed register
    normal! gvy
    " Use the system() function to call sha256sum with the yanked text
    let l:hash = system('echo ' . shellescape(@@) . ' | sha256sum')
    " Yank the hash into Vim's unnamed register
    let @" = l:hash
endfunction
```

4 Data Structures

4.1 2D Segment Tree

4.1.1 Point update query sum

```
#include "../../Contest/template.cpp"
template <typename T, auto op>
struct SegmentTree2D {
    int h, w;
    vector<vector<T>> t;
    SegmentTree2D(const vector<vector<T>> &a)
        : h(a.size()), w(a.back().size()), t(h * 4, vector<T>(w * 4)) {
        build_x(1, 0, h - 1, a);
    }
    void build_y(int vx, int lx, int rx, int vy, int ly, int ry,
                 const vector<vector<T>> &a) {
        if (ly == ry) {
            if (lx == rx)
                t[vx][vy] = a[lx][ly];
            else
                t[vx][vy] = op(t[vx * 2][vy], t[vx * 2 + 1][vy]);
        } else {
            int my = (ly + ry) / 2;
            build_y(vx, lx, rx, vy * 2, ly, my, a);
            build_y(vx, lx, rx, vy * 2 + 1, my + 1, ry, a);
            t[vx][vy] = op(t[vx][vy * 2], t[vx][vy * 2 + 1]);
        }
    }
    void build_x(int vx, int lx, int rx, const vector<vector<T>> &a) {
        if (lx != rx) {
            int mx = (lx + rx) / 2;
            build_x(vx * 2, lx, mx, a);
            build_x(vx * 2 + 1, mx + 1, rx, a);
        }
        build_y(vx, lx, rx, 1, 0, w - 1, a);
    }
    T query_y(int vx, int vy, int tly, int try_, int ly, int ry) {
        if (ly > ry) return 0;
        if (ly == tly && try_ == ry) return t[vx][vy];
        int tmy = (tly + try_) / 2;
        return op(query_y(vx, vy * 2, tly, tmy, ly, min(ry, tmy)),
                  query_y(vx, vy * 2 + 1, tmy + 1, try_, max(ly, tmy + 1),
 ry));
    }
    T query_x(int vx, int tlx, int trx, int lx, int rx, int ly, int ry) {
        if (lx > rx) return 0;
        if (lx == tlx && trx == rx) return query_y(vx, 1, 0, w - 1, ly, ry
);
        int tmx = (tlx + trx) / 2;
        return op(
            query_x(vx * 2, tlx, tmx, lx, min(rx, tmx), ly, ry),
            query_x(vx * 2 + 1, tmx + 1, trx, max(lx, tmx + 1), rx, ly, ry
));
    }
    void update_y(int vx, int lx, int rx, int vy, int ly, int ry, int x,
int y,
                  int new_val) {
        if (ly == ry) {
            if (lx == rx)
                t[vx][vy] = new_val;
            else
                t[vx][vy] = op(t[vx * 2][vy], t[vx * 2 + 1][vy]);
        } else {
            int my = (ly + ry) / 2;
            if (y <= my)
                update_y(vx, lx, rx, vy * 2, ly, my, x, y, new_val);
            else
                update_y(vx, lx, rx, vy * 2 + 1, my + 1, ry, x, y, new_val
);
            t[vx][vy] = op(t[vx][vy * 2], t[vx][vy * 2 + 1]);
        }
    }
    void update_x(int vx, int lx, int rx, int x, int y, T new_val) {
        if (lx != rx) {
            int mx = (lx + rx) / 2;
            if (x <= mx)
                update_x(vx * 2, lx, mx, x, y, new_val);
            else
                update_x(vx * 2 + 1, mx + 1, rx, x, y, new_val);
        }
        update_y(vx, lx, rx, 1, 0, w - 1, x, y, new_val);
    }
    T query(int lx, int rx, int ly, int ry) {
        return query_x(1, 0, h - 1, lx, rx, ly, ry);
    }
};
```

4.2 SQRT decomposition

4.2.1 two-sequence-queries

```
using ll = long long;
const ll MOD = 998244353;
inline ll sum(const ll a, const ll b) { return (a + b) % MOD; }
ll sub(const ll a, const ll b) { return (a - b + MOD) % MOD; }
inline ll mul(const ll a, const ll b) { return (a * b) % MOD; }
struct SqrtDecomposition {
    struct t_sqrt {
        int l, r;
        ll x, y;
        ll prod;
        ll sum_as, sum_bs;
        t_sqrt() {
            l = numeric_limits<int>::max();
```

```
        r = numeric_limits<int>::min();
        x = y = prod = sum_as = sum_bs = 0;
    };
};
int sqrtLen;
vector<t_sqrt> blocks;
vector<ll> as, bs;
SqrtDecomposition(const vector<ll> &as_, const vector<ll> &bs_) {
    int n = as_.size();
    sqrtLen = (int)sqrt(n + .0) + 1;
    blocks.resize(sqrtLen + 6.66);
    as = as_;
    bs = bs_;
    for (int i = 0; i < n; i++) {
        auto &bi = blocks[i / sqrtLen];
        bi.l = min(bi.l, i);
        bi.r = max(bi.r, i);
        bi.sum_as = sum(bi.sum_as, as[i]);
        bi.sum_bs = sum(bi.sum_bs, bs[i]);
        bi.prod = sum(bi.prod, mul(as[i], bs[i]));
    }
}
// adds x to a[i], and y to b[i], in range [l,
// r]
void update(int l, int r, ll x, ll y) {
    auto apply1 = [&](int idx, ll x, ll y) -> void {
        auto &block = blocks[idx / sqrtLen];
        block.prod = sub(block.prod, mul(as[idx], bs[idx]));
        block.sum_as = sub(block.sum_as, as[idx]);
        block.sum_bs = sub(block.sum_bs, bs[idx]);
        as[idx] = sum(as[idx], x);
        bs[idx] = sum(bs[idx], y);
        block.prod = sum(block.prod, as[idx] * bs[idx]);
        block.sum_as = sum(block.sum_as, as[idx]);
        block.sum_bs = sum(block.sum_bs, bs[idx]);
    };
    auto apply2 = [&](int idx, ll x, ll y) -> void {
        blocks[idx].x = sum(blocks[idx].x, x);
        blocks[idx].y = sum(blocks[idx].y, y);
    };
    int cl = l / sqrtLen, cr = r / sqrtLen;
    if (cl == cr) {
        for (int i = l; i <= r; i++) {
            apply1(i, x, y);
        }
    } else {
        for (int i = l; i <= (cl + 1) * sqrtLen - 1; i++) {
            apply1(i, x, y);
        }
        for (int i = cl + 1; i <= cr - 1; i++) {
            apply2(i, x, y);
        }
        for (int i = cr * sqrtLen; i <= r; i++) {
            apply1(i, x, y);
        }
    }
}
// sum of a[i]*b[i] in range [l  r]
ll query(int l, int r) {
    auto eval1 = [&](int idx) -> ll {
        auto &block = blocks[idx / sqrtLen];
        return mul(sum(as[idx], +block.x), sum(bs[idx], block.y));
    };
    auto eval2 = [&](int idx) -> ll {
        auto &block = blocks[idx];
        ll ret = 0;
        ret = sum(
            ret, mul(mul(block.x, block.y), sum(sub(block.r, block.l),
1)));
        ret = sum(ret, block.prod);
        ret = sum(ret, block.y * block.sum_as);
        ret = sum(ret, block.x * block.sum_bs);
        return ret;
    };
    ll ret = 0;
    int cl = l / sqrtLen, cr = r / sqrtLen;
    if (cl == cr) {
        for (int i = l; i <= r; i++) {
            ret = sum(ret, eval1(i));
        }
    } else {
        for (int i = l; i <= (cl + 1) * sqrtLen - 1; i++) {
            ret = sum(eval1(i), ret);
        }
        for (int i = cl + 1; i <= cr - 1; i++) {
            ret = sum(ret, eval2(i));
        }
        for (int i = cr * sqrtLen; i <= r; i++) {
            ret = sum(ret, eval1(i));
        }
    }
    return ret;
}
};
```

4.3 Segment Tree Point Update Range Query (bottom-up)

4.3.1 Query GCD

```
using ll = long long;
struct Node {
    ll value;
    bool undef;
    Node() : value(1), undef(1) {};  // Neutral element
```

```cpp
    Node(ll v) : value(v), undef(0) {};
};
inline Node combine(const Node &nl, const Node &nr) {
    if (nl.undef) return nr;
    if (nr.undef) return nl;
    Node m;
    m.value = gcd(nl.value, nr.value);
    m.undef = false;
    return m;
}
template <typename T = Node, auto F = combine>
struct SegTree {
    int n;
    vector<T> st;
    SegTree(int _n) : n(_n), st(n << 1) {}
    void assign(int p, const T &k) {
        for (st[p += n] = k; p >>= 1;) st[p] = F(st[p << 1], st[p << 1 |
    1]);
    }
    T query(int l, int r) {
        T ansl, ansr;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) ansl = F(ansl, st[l++]);
            if (r & 1) ansr = F(st[--r], ansr);
        }
        return F(ansl, ansr);
    }
};
```

4.3.2 Query Max Subarray Sum

```cpp
#pragma once
#include "../../Contest/template.cpp"
#include "./Struct.cpp"
const ll _oo = 1e9;
struct Node {
    ll tot, suf, pref, best;
    // Neutral element
    Node() : tot(-_oo), suf(-_oo), pref(-_oo), best(-_oo) {}  // Neutral
    element
    // for assign
    Node(ll x) { tot = x, suf = x, pref = x, best = max(0ll, x); }
};
Node combine(Node &nl, Node &nr) {
    if (nl.tot == -_oo) return nr;
    if (nr.tot == -_oo) return nl;
    Node m;
    m.tot = nl.tot + nr.tot;
    m.pref = max({nl.pref, nl.tot + nr.pref});
    m.suf = max({nr.suf, nr.tot + nl.suf});
    m.best = max({nl.best, nr.best, nl.suf + nr.pref});
    return m;
}
using SegTreeMaxSubarraySum = SegTreeBottomUp<Node, Node(), combine>;
```

4.3.3 Query min

```cpp
#pragma once
#include "../../Contest/template.cpp"
#include "./Struct.cpp"
template <typename T>
using SegTreeBottomUpMinQuery =
    SegTreeBottomUp<T, numeric_limits<T>::max(),
                    [](T a, T b) { return min(a, b); }>;
```

4.3.4 Query sum

```cpp
#pragma once
#include "../../Contest/template.cpp"
#include "./Struct.cpp"
template <typename T>
using SegTreeBottomUpSumQuery =
    SegTreeBottomUp<T, T(0), [](T a, T b) { return a + b; }>;
```

4.3.5 Struct

```cpp
/*
 * @Description:
 *      merge should be function<T(T,T)>, that
 *      makes the necessary operation between two
 *      nodes in the segment tree
 * */
#pragma once
#include "../../Contest/template.cpp"
template <typename T, T identity, auto merge>
struct SegTreeBottomUp {
    int size;
    vector<T> arr;
    SegTreeBottomUp(int n) {
        for (size = 1; size < n; size <<= 1);
        arr.resize(size << 1);
    }
    void assign(int pos, const T &val) {
        for (arr[pos += size] = val; pos >>= 1;)
            arr[pos] = merge(arr[pos << 1], arr[pos << 1 | 1]);
    }
    T query(int l, int r) {
        T ans_l = identity, ans_r = identity;
        for (l += size, r += size + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) ans_l = merge(ans_l, arr[l++]);
            if (r & 1) ans_r = merge(arr[--r], ans_r);
        }
```

```
        return merge(ans_l, ans_r);
    }
    SegTreeBottomUp(const vector<T> &vec) : SegTreeBottomUp(len(vec)) {
        copy(all(vec), begin(arr) + size);
        rrep(i, size - 1, 0) arr[i] = merge(arr[i << 1], arr[i << 1 | 1]);
    }
};
```

4.4 Segment tree (dynamic)

4.4.1 Range Max Query Point Max Assignment

Description: Answers range queries in ranges until 10^9 (maybe more)
Time: Query and update $O(n \cdot \log n)$

```
struct node;
node *newNode();
struct node {
    node *left, *right;
    int lv, rv;
    ll val;
    node() : left(NULL), right(NULL), val(-oo) {}
    inline void init(int l, int r) {
        lv = l;
        rv = r;
    }
    inline void extend() {
        if (!left) {
            int m = (lv + rv) / 2;
            left = newNode();
            right = newNode();
            left->init(lv, m);
            right->init(m + 1, rv);
        }
    }
    ll query(int l, int r) {
        if (r < lv || rv < l) {
            return 0;
        }
        if (l <= lv && rv <= r) {
            return val;
        }
        extend();
        return max(left->query(l, r), right->query(l, r));
    }
    void update(int p, ll newVal) {
        if (lv == rv) {
            val = max(val, newVal);
            return;
        }
        extend();
        (p <= left->rv ? left : right)->update(p, newVal);
        val = max(left->val, right->val);
    }
};
const int BUFFSZ(1e7);
node *newNode() {
    static int bufSize = BUFFSZ;
    static node buf[(int)BUFFSZ];
    assert(bufSize);
    return &buf[--bufSize];
}
struct SegTree {
    int n;
    node *root;
    SegTree(int _n) : n(_n) {
        root = newNode();
        root->init(0, n);
    }
    ll query(int l, int r) { return root->query(l, r); }
    void update(int p, ll v) { root->update(p, v); }
};
```

4.4.2 Range Sum Query Point Sum Update

Description: Answers range queries in ranges until 10^9 (maybe more)
Time: Query and update in $O(n \cdot \log n)$

```
struct node;
node *newNode();
struct node {
    node *left, *right;
    int lv, rv;
    ll val;
    node() : left(NULL), right(NULL), val(0) {}
    inline void init(int l, int r) {
        lv = l;
        rv = r;
    }
    inline void extend() {
        if (!left) {
            int m = (rv - lv) / 2 + lv;
            left = newNode();
            right = newNode();
            left->init(lv, m);
            right->init(m + 1, rv);
        }
    }
    ll query(int l, int r) {
        if (r < lv || rv < l) {
            return 0;
        }
        if (l <= lv && rv <= r) {
            return val;
        }
        extend();
```

```
        return left->query(l, r) + right->query(l, r);
    }
    void update(int p, ll newVal) {
        if (lv == rv) {
            val += newVal;
            return;
        }
        extend();
        (p <= left->rv ? left : right)->update(p, newVal);
        val = left->val + right->val;
    }
};
const int BUFFSZ(1.3e7);
node *newNode() {
    static int bufSize = BUFFSZ;
    static node buf[(int)BUFFSZ];
    // assert(bufSize);
    return &buf[--bufSize];
}
struct SegTree {
    int n;
    node *root;
    SegTree(int _n) : n(_n) {
        root = newNode();
        root->init(0, n);
    }
    ll query(int l, int r) { return root->query(l, r); }
    void update(int p, ll v) { root->update(p, v); }
};
```

4.5 Segment tree point update range query (top-down)

4.5.1 Query hash (top down)

```
#include "../../Contest/template.cpp"
const ll MOD = 1'000'000'009;
const ll P = 31;
const int MAXN = 2'000'000;
ll pows[MAXN + 1];
void computepows() {
    pows[0] = 1;
    for (int i = 1; i <= MAXN; i++) {
        pows[i] = (pows[i - 1] * P) % MOD;
    }
}
struct Node {
    ll hash;
    Node() : hash(-1) {};  // Neutral element
    Node(ll v) : hash(v) {};
};
inline Node combine(Node &vl, Node &vr, int nl, int nr, int ql, int qr) {
    if (vl.hash == -1) return vr;
    if (vr.hash == -1) return vl;
    Node vm;
    int nm = midpoint(nl, nr);
    int lsize = min(nm, qr) - max(nl, ql) + 1;
    vm.hash = (vl.hash + ((vr.hash * pows[lsize]) % MOD)) % MOD;
    return vm;
}
template <typename T = Node, auto F = combine>
struct SegTree {
    int n;
    vector<T> st;
    SegTree(int n) : n(n), st(n << 2) {}
    void assign(int p, const T &v) { assign(1, 0, n - 1, p, v); }
    void assign(int node, int l, int r, int p, const T &v) {
        if (l == r) {
            st[node] = v;
            return;
        }
        int m = midpoint(l, r);
        if (p <= m)
            assign(node << 1, l, m, p, v);
        else
            assign(node << 1 | 1, m + 1, r, p, v);
        st[node] = F(st[node << 1], st[node << 1 | 1], l, r, l, r);
    }
    inline T query(int l, int r) { return query(1, 0, n - 1, l, r); }
    inline T query(int node, int nl, int nr, int l, int r) const {
        if (r < nl or nr < l) return T();
        if (l <= nl and nr <= r) return st[node];
        int m = midpoint(nl, nr);
        auto a = query(node << 1, nl, m, l, r);
        auto b = query(node << 1 | 1, m + 1, nr, l, r);
        return F(a, b, nl, nr, l, r);
    }
};
```

4.6 Segment tree range update range query

4.6.1 Arithmetic progression sum update, query sum

Description: Makes arithmetic progression updates in range and sum queries.
Usage: Considering $PA(A, R) = [A + R, A + 2R, A + 3R, ...]$

- **update_set(l, r, A, R):** sets [l, r] to PA(A, R)
- **update_add(l, r, A, R):** sum PA(A, R) in [l, r]
- **query(l, r):** sum in range [l, r]

Time: build $O(N)$, updates and queries $O(logN)$

```
const ll oo = 1e18;
struct SegTree {
    struct Data {
        ll sum;
        ll set_a, set_r, add_a, add_r;
        Data() : sum(0), set_a(oo), set_r(0), add_a(0), add_r(0) {}
```

```
};
int n;
vector<Data> seg;
SegTree(int n_) : n(n_), seg(vector<Data>(4 * n)) {}
void prop(int p, int l, int r) {
    int sz = r - l + 1;
    ll &sum = seg[p].sum, &set_a = seg[p].set_a, &set_r = seg[p].set_r
,
       &add_a = seg[p].add_a, &add_r = seg[p].add_r;
    if (set_a != oo) {
        set_a += add_a, set_r += add_r;
        sum = set_a * sz + set_r * sz * (sz + 1) / 2;
        if (l != r) {
            int m = (l + r) / 2;
            seg[2 * p].set_a = set_a;
            seg[2 * p].set_r = set_r;
            seg[2 * p].add_a = seg[2 * p].add_r = 0;
            seg[2 * p + 1].set_a = set_a + set_r * (m - l + 1);
            seg[2 * p + 1].set_r = set_r;
            seg[2 * p + 1].add_a = seg[2 * p + 1].add_r = 0;
        }
        set_a = oo, set_r = 0;
        add_a = add_r = 0;
    } else if (add_a or add_r) {
        sum += add_a * sz + add_r * sz * (sz + 1) / 2;
        if (l != r) {
            int m = (l + r) / 2;
            seg[2 * p].add_a += add_a;
            seg[2 * p].add_r += add_r;
            seg[2 * p + 1].add_a += add_a + add_r * (m - l + 1);
            seg[2 * p + 1].add_r += add_r;
        }
        add_a = add_r = 0;
    }
}
int inter(pii a, pii b) {
    if (a.first > b.first) swap(a, b);
    return max(0, min(a.second, b.second) - b.first + 1);
}
ll set(int a, int b, ll aa, ll rr, int p, int l, int r) {
    prop(p, l, r);
    if (b < l or r < a) return seg[p].sum;
    if (a <= l and r <= b) {
        seg[p].set_a = aa;
        seg[p].set_r = rr;
        prop(p, l, r);
        return seg[p].sum;
    }
    int m = (l + r) / 2;
    int tam_l = inter({l, m}, {a, b});
    return seg[p].sum = set(a, b, aa, rr, 2 * p, l, m) +
                        set(a, b, aa + rr * tam_l, rr, 2 * p + 1, m +
1, r);
}
    void update_set(int l, int r, ll aa, ll rr) {
        set(l, r, aa, rr, 1, 0, n - 1);
    }
    ll add(int a, int b, ll aa, ll rr, int p, int l, int r) {
        prop(p, l, r);
        if (b < l or r < a) return seg[p].sum;
        if (a <= l and r <= b) {
            seg[p].add_a += aa;
            seg[p].add_r += rr;
            prop(p, l, r);
            return seg[p].sum;
        }
        int m = (l + r) / 2;
        int tam_l = inter({l, m}, {a, b});
        return seg[p].sum = add(a, b, aa, rr, 2 * p, l, m) +
                            add(a, b, aa + rr * tam_l, rr, 2 * p + 1, m +
1, r);
    }
    void update_add(int l, int r, ll aa, ll rr) {
        add(l, r, aa, rr, 1, 0, n - 1);
    }
    ll query(int a, int b, int p, int l, int r) {
        prop(p, l, r);
        if (b < l or r < a) return 0;
        if (a <= l and r <= b) return seg[p].sum;
        int m = (l + r) / 2;
        return query(a, b, 2 * p, l, m) + query(a, b, 2 * p + 1, m + 1, r)
;
    }
    ll query(int l, int r) { return query(l, r, 1, 0, n - 1); }
};
```

4.6.2 Increment update query min & max (bottom up)

```
using SegT = ll;
struct QueryT {
    SegT mx, mn;
    QueryT()
        : mx(numeric_limits<SegT>::min()), mn(numeric_limits<SegT>::max())
    {}
    QueryT(SegT _v) : mx(_v), mn(_v) {}
};
inline QueryT combine(QueryT ln, QueryT rn, pii lr1, pii lr2) {
    chmax(ln.mx, rn.mx);
    chmin(ln.mn, rn.mn);
    return ln;
}
using LazyT = SegT;
inline QueryT applyLazyInQuery(QueryT q, LazyT l, pii lr) {
    if (q.mx == QueryT().mx) q.mx = SegT();
    if (q.mn == QueryT().mn) q.mn = SegT();
    q.mx += l, q.mn += l;
    return q;
```

```
}
inline LazyT applyLazyInLazy(LazyT a, LazyT b) { return a + b; }
using UpdateT = SegT;
inline QueryT applyUpdateInQuery(QueryT q, UpdateT u, pii lr) {
    if (q.mx == QueryT().mx) q.mx = SegT();
    if (q.mn == QueryT().mn) q.mn = SegT();
    q.mx += u, q.mn += u;
    return q;
}
inline LazyT applyUpdateInLazy(LazyT l, UpdateT u, pii lr) { return l + u;
 }
template <typename Qt = QueryT, typename Lt = LazyT, typename Ut = UpdateT
    ,
          auto C = combine, auto ALQ = applyLazyInQuery,
          auto ALL = applyLazyInLazy, auto AUQ = applyUpdateInQuery,
          auto AUL = applyUpdateInLazy>
struct LazySegmentTree {
    int n, h;
    vector<Qt> ts;
    vector<Lt> ds;
    vector<pii> lrs;
    LazySegmentTree(int _n)
        : n(_n),
          h(sizeof(int) * 8 - __builtin_clz(n)),
          ts(n << 1),
          ds(n),
          lrs(n << 1) {
        rep(i, 0, n) lrs[i + n] = {i, i};
        rrep(i, n - 1, 0) {
            lrs[i] = {lrs[i << 1].first, lrs[i << 1 | 1].second};
        }
    }
    LazySegmentTree(const vector<Qt> &xs) : LazySegmentTree(len(xs)) {
        copy(all(xs), ts.begin() + n);
        rep(i, 0, n) lrs[i + n] = {i, i};
        rrep(i, n - 1, 0) {
            ts[i] = C(ts[i << 1], ts[i << 1 | 1], lrs[i << 1], lrs[i << 1
    | 1]);
        }
    }
    void set(int p, Qt v) {
        ts[p + n] = v;
        build(p + n);
    }
    void upd(int l, int r, Ut v) {
        l += n, r += n + 1;
        int l0 = l, r0 = r;
        for (; l < r; l >>= 1, r >>= 1) {
            if (l & 1) apply(l++, v);
            if (r & 1) apply(--r, v);
        }
        build(l0), build(r0 - 1);
    }
    Qt qry(int l, int r) {
        l += n, r += n + 1;
        push(l), push(r - 1);
        Qt resl = Qt(), resr = Qt();
        pii lr1 = {l, l}, lr2 = {r, r};
        for (; l < r; l >>= 1, r >>= 1) {
            if (l & 1) resl = C(resl, ts[l], lr1, lrs[l]), l++;
            if (r & 1) r--, resr = C(ts[r], resr, lrs[r], lr2);
        }
        return C(resl, resr, lr1, lr2);
    }
    void build(int p) {
        while (p > 1) {
            p >>= 1;
            ts[p] =
                ALQ(C(ts[p << 1], ts[p << 1 | 1], lrs[p << 1], lrs[p << 1
| 1]),
                    ds[p], lrs[p]);
        }
    }
    void push(int p) {
        rrep(s, h, 0) {
            int i = p >> s;
            if (ds[i] != Lt()) {
                apply(i << 1, ds[i]), apply(i << 1 | 1, ds[i]);
                ds[i] = Lt();
            }
        }
    }
    inline void apply(int p, Ut v) {
        ts[p] = AUQ(ts[p], v, lrs[p]);
        if (p < n) ds[p] = AUL(ds[p], v, lrs[p]);
    }
};
```

4.6.3 Increment update sum query (top down)

```
struct Lnode {
    ll v;
    bool assign;
    Lnode() : v(), assign() {}  // Neutral element
    Lnode(ll _v, bool a = 0) : v(_v), assign(a) {};
};
using Qnode = ll;
using Unode = Lnode;
struct LSegTree {
    int n, ql, qr;
    vector<Qnode> st;
    vector<Lnode> lz;
    /*-----------------------------------------*/
    Qnode merge(Qnode lv, Qnode rv, int nl, int nr) { return lv + rv; }
    void prop(int i, int l, int r) {
        if (lz[i].assign) {
```

```cpp
        st[i] = lz[i].v * (r - l + 1);
        if (l != r) lz[tol(i)] = lz[tor(i)] = lz[i];
    } else {
        st[i] += lz[i].v * (r - l + 1);
        if (l != r) lz[tol(i)].v += lz[i].v, lz[tor(i)].v += lz[i].v;
    }
    lz[i] = Lnode();
}
void applyV(int i, Unode v) {
    if (v.assign) {
        lz[i] = v;
    } else {
        lz[i].v += v.v;
    }
}
/*-------------------------------------------*/
LSegTree() {}
LSegTree(int _n) : n(_n), st(_n << 2), lz(_n << 2) {}
bool disjoint(int l, int r) { return qr < l or r < ql; }
bool contains(int l, int r) { return ql <= l and r <= qr; }
int tol(int i) { return i << 1; }
int tor(int i) { return i << 1 | 1; }
void build(vector<Qnode> &v) { build(v, 1, 0, n - 1); }
void build(vector<Qnode> &v, int i, int l, int r) {
    if (l == r) {
        st[i] = v[l];
        return;
    }
    int m = midpoint(l, r);
    build(v, tol(i), l, m);
    build(v, tor(i), m + 1, r);
    st[i] = merge(st[tol(i)], st[tor(i)], l, r);
}
void upd(int l, int r, Unode v) {
    ql = l, qr = r;
    upd(1, 0, n - 1, v);
}
void upd(int i, int l, int r, Unode v) {
    prop(i, l, r);
    if (disjoint(l, r)) return;
    if (contains(l, r)) {
        applyV(i, v);
        prop(i, l, r);
        return;
    }
    int m = midpoint(l, r);
    upd(tol(i), l, m, v);
    upd(tor(i), m + 1, r, v);
    st[i] = merge(st[tol(i)], st[tor(i)], l, r);
}
Qnode qry(int l, int r) {
    ql = l, qr = r;
    return qry(1, 0, n - 1);
}
Qnode qry(int i, int l, int r) {
    prop(i, l, r);
        if (disjoint(l, r)) return Qnode();
        if (contains(l, r)) return st[i];
        int m = midpoint(l, r);
        return merge(qry(tol(i), l, m), qry(tor(i), m + 1, r), l, r);
    }
};
```

4.7 2D Sparse Table

```cpp
const int N = 1001;
ll matrix[N][N];
ll M[1001][1001][10][10];
ll op(ll a, ll b) { return gcd(a, b); }
void SparseMatrix(int n, int m) {
    int i, j, x, y;
    for (i = 0; (1 << i) <= n; i++) {
        for (j = 0; (1 << j) <= m; j++) {
            for (x = 0; (x + (1 << i) - 1) < n; x++) {
                for (y = 0; (y + (1 << j) - 1) < m; y++) {
                    if (i == 0 && j == 0)
                        M[x][y][i][j] = matrix[x][y];
                    else if (i == 0)
                        M[x][y][i][j] = op(M[x][y][i][j - 1],
                                           M[x][y + (1 << (j - 1))][i][j -
    1]);
                    else if (j == 0)
                        M[x][y][i][j] = op(M[x][y][i - 1][j],
                                           M[x + (1 << (i - 1))][y][i -
    1][j]);
                    else {
                        int tempa = op(M[x + (1 << (i - 1))][y][i - 1][j -
    1],
                                       M[x][y + (1 << (j - 1))][i - 1][j -
    1]);
                        int tempb = op(M[x][y][i - 1][j - 1],
                                       M[x + (1 << (i - 1))][y + (1 << (j
    - 1))]
                                        [i - 1][j - 1]);
                        M[x][y][i][j] = op(tempa, tempb);
                    }
                }
            }
        }
    }
    return;
}
int lg2(int x) { return sizeof(int) * 8 - __builtin_clz(x) - 1; }
ll query2d(int x, int y, int x1, int y1) {
    int k = lg2(x1 - x + 1);
    int l = lg2(y1 - y + 1);
    int tempa = op(M[x][y][k][l], M[x1 - (1 << k) + 1][y][k][l]);
    int tempb = op(M[x][y1 - (1 << l) + 1][k][l],
                   M[x1 - (1 << k) + 1][y1 - (1 << l) + 1][k][l]);
    return op(tempa, tempb);
```

```
}
```

4.8 Bitree 2D

Description: Given a 2D array you can increment an arbitrary position, and also query the subsum of a subgrid
Time: Update and query in $O(logN^2)$

```
struct Bit2d {
    int n;
    vll2d bit;
    Bit2d(int ni) : n(ni), bit(n + 1, vll(n + 1)) {}
    Bit2d(int ni, vll2d &xs) : n(ni), bit(n + 1, vll(n + 1)) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                update(i, j, xs[i][j]);
            }
        }
    }
    void update(int x, int y, ll val) {
        for (; x <= n; x += (x & (-x))) {
            for (int i = y; i <= n; i += (i & (-i))) {
                bit[x][i] += val;
            }
        }
    }
    ll sum(int x, int y) {
        ll ans = 0;
        for (int i = x; i; i -= (i & (-i))) {
            for (int j = y; j; j -= (j & (-j))) {
                ans += bit[i][j];
            }
        }
        return ans;
    }
    ll query(int x1, int y1, int x2, int y2) {
        return sum(x2, y2) - sum(x2, y1 - 1) - sum(x1 - 1, y2) +
               sum(x1 - 1, y1 - 1);
    }
};
```

4.9 Convex Hull Trick / Line Container

Description: Container where you can add lines of the form $mx + b$, and query the maximum value at point x.
Usage: $insert\_line(m, b)$ inserts the line $m \cdot x + b$ in the container.
$eval(x)$ find the highest value among all lines in the point x.
Time: Eval and insert in $O(\log N)$

```
const ll LLINF = 1e18;
const ll is_query = -LLINF;
struct Line {
    ll m, b;
    mutable function<const Line *()> succ;
    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct Cht : public multiset<Line> {  // maintain
                                      // max m*x+b
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (ld)(x->b - y->b) * (z->m - y->m) >=
               (ld)(y->b - z->b) * (y->m - x->m);
    }
    void insert_line(ll m,
                     ll b) {  // min -> insert (-m,-b) -> -eval()
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line){x, is_query});
        return l.m * x + l.b;
    }
};
```

4.10 DSU (with rollback)

Description: Performs every operation a regular DSU does, but you can roll back to a specific time.
Usage: int t = uf.time(); ...; uf.rollback(t); T
Time: $O(\log(N))$

```
struct RollbackUF {
    vi e;
    vector<pii> st;
    RollbackUF(int n) : e(n, -1) {}
    int size(int x) { return -e[find(x)]; }
    int find(int x) { return e[x] < 0 ? x : find(e[x]); }
    int time() { return len(st); }
    void rollback(int t) {
        for (int i = time(); i-- > t;) e[st[i].first] = st[i].second;
        st.resize(t);
    }
```

```cpp
    bool join(int a, int b) {
        a = find(a), b = find(b);
        if (a == b) return false;
        if (e[a] > e[b]) swap(a, b);
        st.push_back({a, e[a]});
        st.push_back({b, e[b]});
        e[a] += e[b];
        e[b] = a;
        return true;
    }
};
```

4.11 DSU / UFDS

Usage: You may discomment the commented parts to find online which nodes belong to each set, it makes the *union_set* method cost $O(log^2)$ instead $O(A)$

```cpp
struct DSU {
    vector<int> ps, sz;
    // vector<unordered_set<int>> sts;
    DSU(int N)
        : ps(N + 1),
          sz(N, 1) /*, sts(N) */
    {
        iota(ps.begin(), ps.end(), 0);
        // for (int i = 0; i < N; i++)
        // sts[i].insert(i);
    }
    int find_set(int x) { return ps[x] == x ? x : ps[x] = find_set(ps[x]);
 }
    int size(int u) { return sz[find_set(u)]; }
    bool same_set(int x, int y) { return find_set(x) == find_set(y); }
    void union_set(int x, int y) {
        if (same_set(x, y)) return;
        int px = find_set(x);
        int py = find_set(y);
        if (sz[px] < sz[py]) swap(px, py);
        ps[py] = px;
        sz[px] += sz[py];
        // sts[px].merge(sts[py]);
    }
};
```

4.12 Lichao Tree (dynamic)

Description: Lichao Tree that creates the nodes dynamically, allowing to query and update from range $[MAXL, MAXR]$

Usage:

- $query(x)$: find the highest point among all lines in the structure
- $add(a, b)$: add a line of form $y = ax + b$ in the structure
- $addSegment(a, b, l, r)$: add a line segment of form $y = ax + b$ which covers from range $[l, r]$

Time: $O(\log N)$

```cpp
template <typename T = ll, T MAXL = 0, T MAXR = 1 '000' 000'001>
struct LiChaoTree {
    static const T inf = -numeric_limits<T>::max() / 2;
    bool first_best(T a, T b) { return a > b; }
    T get_best(T a, T b) { return first_best(a, b) ? a : b; }
    struct line {
        T m, b;
        T operator()(T x) { return m * x + b; }
    };
    struct node {
        line li;
        node *left, *right;
        node(line _li = {0, inf}) : li(_li), left(nullptr), right(nullptr)
 {}
        ~node() {
            delete left;
            delete right;
        }
    };
    node *root;
    LiChaoTree(line li = {0, inf}) : root(new node(li)) {}
    ~LiChaoTree() { delete root; }
    T query(T x, node *cur, T l, T r) {
        if (cur == nullptr) return inf;
        if (x < l or x > r) return inf;
        T mid = midpoint(l, r);
        T ans = cur->li(x);
        ans = get_best(ans, query(x, cur->left, l, mid));
        ans = get_best(ans, query(x, cur->right, mid + 1, r));
        return ans;
    }
    T query(T x) { return query(x, root, MAXL, MAXR); }
    void add(line li, node *&cur, T l, T r) {
        if (cur == nullptr) {
            cur = new node(li);
            return;
        }
        T mid = midpoint(l, r);
        if (first_best(li(mid), cur->li(mid))) swap(li, cur->li);
        if (first_best(li(l), cur->li(l))) add(li, cur->left, l, mid);
        if (first_best(li(r), cur->li(r))) add(li, cur->right, mid + 1, r)
;
    }
    void add(T m, T b) { add({m, b}, root, MAXL, MAXR); }
    void addSegment(line li, node *&cur, T l, T r, T lseg, T rseg) {
        if (r < lseg || l > rseg) return;
        if (cur == nullptr) cur = new node;
        if (lseg <= l && r <= rseg) {
            add(li, cur, l, r);
            return;
        }
        T mid = midpoint(l, r);
        if (l != r) {
            addSegment(li, cur->left, l, mid, lseg, rseg);
```

```
        addSegment(li, cur->right, mid + 1, r, lseg, rseg);
      }
  }
  void addSegment(T a, T b, T l, T r) {
      addSegment({a, b}, root, MAXL, MAXR, l, r);
  }
};
```

4.13 Merge sort tree

Description: Like a segment tree but each node stores the ordered subsegment it represents.
Usage:

- $inrange(l, r, a, b)$: counts the number of positions i, $l \leq i \leq r$ such that $a \leq x_i \leq b$.

Time: Build $O(N \log N^2)$, inrange $O(\log N^2)$
Memory: $O(n \log N)$

```
template <class T>
struct MergeSortTree {
    int n;
    vector<vector<T>> st;
    MergeSortTree(vector<T> &xs) : n(len(xs)), st(n << 1) {
        rep(i, 0, n) st[i + n] = vector<T>({xs[i]});
        rrep(i, n - 1, 0) {
            st[i].resize(len(st[i << 1]) + len(st[i << 1 | 1]));
            merge(all(st[i << 1]), all(st[i << 1 | 1]), st[i].begin());
        }
    }
    int count(int i, T a, T b) {
        return upper_bound(all(st[i]), b) - lower_bound(all(st[i]), a);
    }
    int inrange(int l, int r, T a, T b) {
        int ans = 0;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) ans += count(l++, a, b);
            if (r & 1) ans += count(--r, a, b);
        }
        return ans;
    }
};
```

4.14 Mex with update

Description: This DS allows you to mantain an array of elments, insert, and remove, and query the MEX at any time.
Usage:

- $Mex(mxsz)$: Initialize the DS, $mxsz$ must be the maximum number of elements that the structure may have.
- $add(x)$: just adds one copy of x.
- $rmv(x)$: just remove a copy of x.
- $operator()$: returns the MEX.

Time:

- $Mex(mxsz)$: $O(\log mxsz)$
- $add(x)$: $O(\log mxsz)$
- $rmv(x)$: $O(\log mxsz)$
- $operator()$: $O(1)$

```
struct Mex {
    int mx_sz;
    vi hs;
    set<int> st;
    Mex(int _mx_sz) : mx_sz(_mx_sz), hs(mx_sz + 1) {
        auto it = st.begin();
        rep(i, 0, mx_sz + 1) it = st.insert(it, i);
    }
    void add(int x) {
        if (x > mx_sz) return;
        if (!hs[x]++) st.erase(x);
    }
    void rmv(int x) {
        if (x > mx_sz) return;
        if (!--hs[x]) st.emplace(x);
    }
    int operator()() const { return *st.begin(); }
    /*
      Optional, you can just create with size
      len(xs) add N elements :D
    */
    Mex(const vi &xs, int _mx_sz = -1) : Mex(~_mx_sz ? _mx_sz : len(xs)) {
        for (auto xi : xs) add(xi);
    }
};
```

4.15 Orderd Set (GNU PBDS)

Usage: If you need an ordered **multi** set you may add an id to each value. Using greater_equal, or less_equal is considered undefined behavior.

- **order_of_key (k)**: Number of items strictly smaller/greater than k .
- **find_by_order(k)**: K-th element in a set (counting from zero).

Time: Both $O(\log N)$

Warning: Is 2 or 3 times slower then a regular set/map.

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T>
using ordered_set =
    tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

4.16 Prefix Sum 2D

Description: Given an 2D array with N lines and M columns, find the sum of the subarray that have the left upper corner at $(x1, y1)$ and right bottom corner at $(x2, y2)$.
Time: Build $O(N \cdot M)$, Query $O(1)$.

```
template <typename T>
struct psum2d {
    vector<vector<T>> s;
    vector<vector<T>> psum;
    psum2d(vector<vector<T>> &grid, int n, int m)
        : s(n + 1, vector<T>(m + 1)), psum(n + 1, vector<T>(m + 1)) {
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++) {
                s[i][j] = s[i][j - 1] + grid[i - 1][j - 1];
                psum[i][j] = psum[i - 1][j] + s[i][j];
            }
    }
    T query(int x1, int y1, int x2, int y2) {
        T ans = psum[x2 + 1][y2 + 1] + psum[x1][y1];
        ans -= psum[x2 + 1][y1] + psum[x1][y2 + 1];
        return ans;
    }
};
```

4.17 Segment Tree Update Range Query (bottom-up)

```
/*
 * @Description:
 *      merge should be function<T(T,T)>, that
 *      makes the necessary operation between two
 *      nodes in the segment tree
 * */
#include "../../Contest/template.cpp"
template <typename T, T identity, auto merge>
struct SegTreeBottomUp {
    int size;
    vector<T> arr;
    SegTreeBottomUp(int n) {
        for (size = 1; size < n; size <<= 1);
        arr.resize(size << 1);
    }
    void assign(int pos, const T &val) {
        for (arr[pos += size] = val; pos >>= 1;)
            arr[pos] = merge(arr[pos << 1], arr[pos << 1 | 1]);
    }
    T query(int l, int r) {
        T ans_l = identity, ans_r = identity;
        for (l += size, r += size + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) ans_l = merge(ans_l, arr[l++]);
            if (r & 1) ans_r = merge(arr[--r], ans_r);
        }
        return merge(ans_l, ans_r);
    }
    SegTreeBottomUp(const vector<T> &vec) : SegTreeBottomUp(len(vec)) {
        copy(all(vec), begin(arr) + size);
        rrep(i, size - 1, 0) arr[i] = merge(arr[i << 1], arr[i << 1 | 1]);
    }
};
using SegTreeBottomUpSumQuery =
    SegTreeBottomUp<ll, 0ll, [](ll a, ll b) { return a + b; }>;
```

4.18 Sparse table

```
template <typename T = ll,
          auto cmp = [](T &src1, T &src2, T &dst) { dst = min(src1, src2);
     }>
class SparseTable {
   private:
    int sz;
    vi logs;
    vector<vector<T>> st;
   public:
    SparseTable(const vector<T> &v) : sz(len(v)), logs(sz + 1) {
        rep(i, 2, sz + 1) logs[i] = logs[i >> 1] + 1;
        st.resize(logs[sz] + 1, vector<T>(sz));
        rep(i, 0, sz) st[0][i] = v[i];
        for (int k = 1; (1 << k) <= sz; k++) {
            for (int i = 0; i + (1 << k) <= sz; i++) {
                cmp(st[k - 1][i], st[k - 1][i + (1 << (k - 1))], st[k][i])
    ;
            }
        }
    }
    T query(int l, int r) {
        r++;
        const int k = logs[r - l];
        T ret;
        cmp(st[k][l], st[k][r - (1 << k)], ret);
        return ret;
    }
};
```

4.19 Static range queries

```
template <typename T = ll,
          auto op = [](const T &src1, const T &src2,
                       T &dst) { dst = src1 + src2; },
          auto invop = [](const T &src1, const T &src2,
                          T &dst) { dst = src1 - src2; }>
struct StaticRangeQueries {
    vector<T> acc;
    StaticRangeQueries(const vector<T> &XS) : acc(len(XS)) {
        acc[0] = XS[0];
        rep(i, 1, len(XS)) { op(acc[i - 1], XS[i], acc[i]); }
```

```
    }
    T operator()(int l, int r) {
        T lv = (l ? acc[l - 1] : T());
        T ret;
        invop(acc[r], lv, ret);
        return ret;
    }
};
```

4.20 Venice Set

Description: A container that you can insert q copies of element e, increment every element in the contaiver in x, query which is the best element and it's quantity and also remove k copies of the greatest element.
Time:

- add elment $O(\log N)$
- remove $O(\log N)$
- update: $O(1)$
- query $O(1)$

```
template <typename T = ll>
struct VeniceSet {
    using T2 = pair<T, ll>;
    priority_queue<T2, vector<T2>, greater<T2>> pq;
    T acc;
    VeniceSet() : acc() {}
    void add_element(const T &e, const ll q) { pq.emplace(e - acc, q); }
    void update_all(const T &x) { acc += x; }
    T2 best() {
        auto ret = pq.top();
        ret.first += acc;
        return ret;
    }
    void pop() { pq.pop(); }
    void pop_k(int k) {
        auto [e, q] = pq.top();
        pq.pop();
        q -= k;
        if (q) pq.emplace(e, q);
    }
};
```

4.21 Venice Set (complete)

Description: A container which you can insert elements update all at once and also make a few queries
Usage:

- $add\_element(e, q)$: adds q copies of e, if no q is provided adds a single one
- $update\_all(x)$: increment every value by x
- $erase(e)$: removes every copy of e, and returns how much was removed.
- $count(e)$: returns the number of e in the container
- $high()/low()$: returns the hightest/lowest element, and it's quantity
- $pop\_low(q)/pop\_high(q)$: removes q copies of the lowest/highest elements if no q is provided removes all copies of the lowest/highest element.

You may answer which is the $K-th$ value and it's quantity using an *ordered_set*.
Probably works with other operations
Time: Considering N the number of distinct numbers in the container

- $add\_element(e, q)$: $O(\log(N))$
- $update\_all(x)$:$O(1)$
- $erase(e)$: $O(\log(N))$
- $count(e)$: $O(\log(N))$
- $high()/low()$: $O(1)$
- $pop\_low(q)/pop\_high(q)$: worst case is $O(N \cdot \log(N))$ if you remove all elements and so on...

Warning: There is no error handling if you try to *pop* more elements than exists or related stuff

---

```
struct VeniceSet {
    set<pll> st;
    ll acc;
    VeniceSet() : acc() {}
    ll add_element(ll e, ll q = 1) {
        q += erase(e);
        e -= acc;
        st.emplace(e, q);
        return q;
    }
    void update_all(ll x) { acc += x; }
    ll erase(ll e) {
        e -= acc;
        auto it = st.lb({e, LLONG_MIN});
        if (it == end(st) || (*it).first != e) return 0;
        ll ret = (*it).second;
        st.erase(it);
        return ret;
    }
    ll count(ll x) {
        x -= acc;
        auto it = st.lb({x, LLONG_MIN});
        if (it == end(st) || (*it).first != x) return 0;
        return (*it).second;
    }
    pll high() { return *rbegin(st); }
    pll low() { return *begin(st); }
    void pop_high(ll q = -1) {
        if (q == -1) q = high().second;
        while (q) {
            auto [e, eq] = high();
            st.erase(prev(end(st)));
            if (eq > q) add_element(e, eq - q);
            q = max(0ll, q - eq);
        }
    }
    void pop_low(ll q = -1) {
```

```
        if (q == -1) q = low().second;
        while (q) {
            auto [e, eq] = low();
            st.erase(st.begin());
            if (eq > q) add_element(e, eq - q);
            q = max(0ll, q - eq);
        }
    }
};
```

4.22 Wavelet tree

```
using ll = long long;
template <typename T>
struct WaveletTree {
    struct Node {
        T lo, hi;
        int left_child, right_child;
        vector<int> pcnt;
        vector<ll> psum;
        Node(int lo_, int hi_)
            : lo(lo_), hi(hi_), left_child(0), right_child(0), pcnt(),
    psum() {}
    };
    vector<Node> nodes;
    WaveletTree(vector<T> v) {
        nodes.reserve(2 * v.size());
        auto [mn, mx] = minmax_element(v.begin(), v.end());
        auto build = [&](auto &&self, Node &node, auto from, auto to) {
            if (node.lo == node.hi or from >= to) return;
            auto mid = midpoint(node.lo, node.hi);
            auto f = [&mid](T x) { return x <= mid; };
            node.pcnt.reserve(to - from + 1);
            node.pcnt.push_back(0);
            node.psum.reserve(to - from + 1);
            node.psum.push_back(0);
            T left_upper = node.lo, right_lower = node.hi;
            for (auto it = from; it != to; it++) {
                auto value = f(*it);
                node.pcnt.push_back(node.pcnt.back() + value);
                node.psum.push_back(node.psum.back() + *it);
                if (value)
                    left_upper = max(left_upper, *it);
                else
                    right_lower = min(right_lower, *it);
            }
            auto pivot = stable_partition(from, to, f);
            node.left_child = make_node(node.lo, left_upper);
            self(self, nodes[node.left_child], from, pivot);
            node.right_child = make_node(right_lower, node.hi);
            self(self, nodes[node.right_child], pivot, to);
        };
        build(build, nodes[make_node(*mn, *mx)], v.begin(), v.end());
    }
    T kth_element(int L, int R, int K) const {
        auto f = [&](auto &&self, const Node &node, int l, int r, int k)
    -> T {
            if (l > r) return 0;
            if (node.lo == node.hi) return node.lo;
            int lb = node.pcnt[l], rb = node.pcnt[r + 1], left_size = rb -
     lb;
            return (left_size > k
                        ? self(self, nodes[node.left_child], lb, rb - 1, k
    )
                        : self(self, nodes[node.right_child], l - lb, r -
    rb,
                               k - left_size));
        };
        return f(f, nodes[0], L, R, K);
    }
    pair<int, ll> count_and_sum_in_range(int L, int R, T a, T b) const {
        auto f = [&](auto &&self, const Node &node, int l,
                     int r) -> pair<int, ll> {
            if (l > r or node.lo > b or node.hi < a) return {0, 0};
            if (a <= node.lo and node.hi <= b)
                return {r - l + 1,
                        (node.lo == node.hi ? (r - l + 1ll) * node.lo
                                            : node.psum[r + 1] - node.psum
    [l])};
            int lb = node.pcnt[l], rb = node.pcnt[r + 1];
            auto [left_cnt, left_sum] =
                self(self, nodes[node.left_child], lb, rb - 1);
            auto [right_cnt, right_sum] =
                self(self, nodes[node.right_child], l - lb, r - rb);
            return {left_cnt + right_cnt, left_sum + right_sum};
        };
        return f(f, nodes[0], L, R);
    }
    inline int count_in_range(int L, int R, T a, T b) const {
        return count_and_sum_in_range(L, R, a, b).first;
    }
    inline ll sum_in_range(int L, int R, T a, T b) const {
        return count_and_sum_in_range(L, R, a, b).second;
    }
   private:
    int make_node(T lo, T hi) {
        int id = (int)nodes.size();
        nodes.emplace_back(lo, hi);
        return id;
    }
};
```

5 Dynamic Programming

5.1 Binary Knapsack (bottom up)

Description: Given the points each element have, and it repespective cost, computes the maximum points we can get if we can ignore/choose an element, in such way that the sum of costs don't exceed the maximum cost allowed.
Time: $O(N * W)$
Warning: The vectors VS and WS starts at one, so it need an empty value at index 0.

```
const int MAXN(1'000), MAXCOST(1'000 * 20);
ll dp[MAXN + 1][MAXCOST + 1];
bool ps[MAXN + 1][MAXCOST + 1];
pair<ll, vi> knapsack(const vll &points, const vi &costs, int maxCost) {
    int n = len(points) - 1;  // ELEMENTS START AT INDEX 1 !
    for (int m = 0; m <= maxCost; m++) {
        dp[0][m] = 0;
    }
    for (int i = 1; i <= n; i++) {
        dp[i][0] = dp[i - 1][0] + (costs[i] == 0) * points[i];
        ps[i][0] = costs[i] == 0;
    }
    for (int i = 1; i <= n; i++) {
        for (int m = 1; m <= maxCost; m++) {
            dp[i][m] = dp[i - 1][m], ps[i][m] = 0;
            int w = costs[i];
            ll v = points[i];
            if (w <= m and dp[i - 1][m - w] + v > dp[i][m]) {
                dp[i][m] = dp[i - 1][m - w] + v, ps[i][m] = 1;
            }
        }
    }
    vi is;
    for (int i = n, m = maxCost; i >= 1; --i) {
        if (ps[i][m]) {
            is.emplace_back(i);
            m -= costs[i];
        }
    }
    return {dp[n][maxCost], is};
}
```

5.2 Edit Distance

Time: $O(N * M)$

```
#include "../Contest/template.cpp"
ll edit_distance(const string &a, const string &b) {
    int n = a.size();
    int m = b.size();
    vll2d dp(n + 1, vi(m + 1, 0));
    const ll ADD = 1, DEL = 1, CHG = 1;
    for (int i = 0; i <= n; ++i) {
        dp[i][0] = i * DEL;
    }
    for (int i = 1; i <= m; ++i) {
        dp[0][i] = ADD * i;
    }
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int add = dp[i][j - 1] + ADD;
            int del = dp[i - 1][j] + DEL;
            int chg = dp[i - 1][j - 1] + (a[i - 1] != b[j - 1]) * CHG;
            dp[i][j] = min({add, del, chg});
        }
    }
    return dp[n][m];
}
```

5.3 Knapsack

Description: Finds the maximum score you can achieve, given that you have N items, each item has a *cost*, a *point* and a *quantity*, you can spent at most *maxcost* and buy each item the maximum quantity it has.
Time: $O(n \cdot maxcost \cdot \log maxqtd)$
Memory: $O(maxcost)$.

```
ll knapsack(const vi &weight, const vll &value, const vi &qtd, int maxCost
    ) {
    vi costs;
    vll values;
    for (int i = 0; i < len(weight); i++) {
        ll q = qtd[i];
        for (ll x = 1; x <= q; q -= x, x <<= 1) {
            costs.eb(x * weight[i]);
            values.eb(x * value[i]);
        }
        if (q) {
            costs.eb(q * weight[i]);
            values.eb(q * value[i]);
        }
    }
    vll dp(maxCost + 1);
    for (int i = 0; i < len(values); i++) {
        for (int j = maxCost; j > 0; j--) {
            if (j >= costs[i]) dp[j] = max(dp[j], values[i] + dp[j - costs
    [i]]);
        }
    }
    return dp[maxCost];
}
```

5.4 Longest Increasing Subsequence

Description: Find the pair (sz, psx) where sz is the size of the longest subsequence and psx is a vector where psx_i tells the size of the longest increase subsequence that ends at

position i. get_idx just tells which indices could be in the longest increasing subsequence.
Time: $O(n \log n)$

```
#include "../Contest/template.cpp"
template <typename T>
pair<int, vi> lis(const vector<T> &xs, int n) {
    vector<T> dp(n + 1, numeric_limits<T>::max());
    dp[0] = numeric_limits<T>::min();
    int sz = 0;
    vi psx(n);
    rep(i, 0, n) {
        int pos = lower_bound(all(dp), xs[i]) - dp.begin();
        sz = max(sz, pos);
        dp[pos] = xs[i];
        psx[i] = pos;
    }
    return {sz, psx};
}
template <typename T>
vi get_idx(vector<T> xs) {
    int n = xs.size();
    auto [sz1, psx1] = lis(xs, n);
    transform(rall(xs), xs.begin(), [](T x) { return -x; });
    auto [sz2, psx2] = lis(xs, n);
    vi ans;
    rep(i, 0, n) {
        int l = psx1[i];
        int r = psx2[n - i - 1];
        if (l + r - 1 == sz1) ans.eb(i);
    }
    return ans;
}
```

5.5 Monery sum

Description: Find every possible sum using the given values only once.

```
set<int> money_sum(const vi &xs) {
    using vc = vector<char>;
    using vvc = vector<vc>;
    int _m = accumulate(all(xs), 0);
    int _n = xs.size();
    vvc _dp(_n + 1, vc(_m + 1, 0));
    set<int> _ans;
    _dp[0][xs[0]] = 1;
    for (int i = 1; i < _n; ++i) {
        for (int j = 0; j <= _m; ++j) {
            if (j == 0 or _dp[i - 1][j]) {
                _dp[i][j + xs[i]] = 1;
                _dp[i][j] = 1;
            }
        }
    }
    for (int i = 0; i < _n; ++i)
        for (int j = 0; j <= _m; ++j)
            if (_dp[i][j]) _ans.insert(j);
    return _ans;
}
```

5.6 Steiner tree

```
template <typename T>
T steinerCost(const vector<vector<T>> &adj, const vi ks,
              T inf = numeric_limits<T>::max()) {
    int k = len(ks), n = len(adj);
    vector<vector<T>> dp(n, vector<T>(1 << k, inf));
    vi inks(n);
    trav(ki, ks) inks[ki] = 1;
    trav(ki, ks) {
        rep(j, 0, n) {
            if (count(all(ks), j) == 0) {
                dp[j][1 << ki] = adj[ki][j];
            }
        }
    }
    rep(mask, 2, (1 << k)) {
        rep(i, 0, n) {
            if (inks[i]) continue;
            for (int mask2 = (mask - 1) & mask; mask2 >= 1;
                 mask2 = (mask2 - 1) & mask) {
                int mask3 = mask ^ mask2;
                chmin(dp[i][mask], dp[i][mask2] + dp[i][mask3]);
            }
            rep(j, 0, n) {
                if (inks[j]) continue;
                chmin(dp[j][mask], dp[i][mask] + adj[i][j]);
            }
        }
    }
    T ans = inf;
    rep(i, 0, n) chmin(ans, dp[i][(1 << k) - 1]);
    return ans;
}
```

5.7 Sum of Subsets

Description: Allows you to find if some mask X is a super mask of any of the given masks
Usage: Call *build* with the *masks* then it returns a vector of bool V where V_X says if X is a super mask of any of the initial maks
You can change it to count how many submasks of each mask exsists, by changing the bitwise or by a plus sign...
Time: $O(LOG \cdot 2^{LOG})$
Memory: $O(LOG^2 \cdot 2^{LOG})$

Warning: Remember to set *LOG* with the highest bit possible

```
const int LOG = 20;
vc build(const vi &masks) {
    vc ret(1 << LOG);
    trav(mi, masks) ret[mi] = 1;
    rep(b, 0, LOG) {
        rep(mask, 0, (1 << LOG)) {
            if (mask & (1 << b)) ret[mask] |= ret[mask ^ (1 << b)];
        }
    }
    return ret;
}
```

5.8 Travelling Salesman Problem

Time: $O(N^2 \cdot 2^N)$
Memory: $O(N^2 \cdot 2^N)$

```
vll2d dist;
vll memo;
int tsp(int i, int mask, int N) {
    if (mask == (1 << N) - 1) return dist[i][0];
    if (memo[i][mask] != -1) return memo[i][mask];
    int ans = INT_MAX << 1;
    for (int j = 0; j < N; ++j) {
        if (mask & (1 << j)) continue;
        auto t = tsp(j, mask | (1 << j), N) + dist[i][j];
        ans = min(ans, t);
    }
    return memo[i][mask] = ans;
}
```

6 Extras

6.1 Binary to gray

```
string binToGray(string bin) {
    string gray(bin.size(), '0');
    int n = bin.size() - 1;
    gray[0] = bin[0];
    for (int i = 1; i <= n; i++) {
        gray[i] = '0' + (bin[i - 1] == '1') ^ (bin[i] == '1');
    }
    return gray;
}
```

6.2 Get permutation cycles

Description: Receives a permutation [0, n-1] and return a vector 2D with each cycle.

```
vll2d getPermutationCicles(const vll &ps) {
    ll n = len(ps);
    vector<char> visited(n);
    vector<vll> cicles;
    rep(i, 0, n) {
        if (visited[i]) continue;
        vll cicle;
        ll pos = i;
        while (!visited[pos]) {
            cicle.pb(pos);
            visited[pos] = true;
            pos = ps[pos];
        }
        cicles.push_back(vll(all(cicle)));
    }
    return cicles;
}
```

6.3 Max & Min Check

Description: Returns the min/max value in range [l, r] that satisfies the lambda function check, if there is no such value the 'nullopt' is returned.

Usage: check must be a function that receives an integer and return a boolean.

Time: $O(\log r - l + 1)$

```
template <typename T>
optional<T> maxCheck(T l, T r, auto check) {
    optional<T> ret;
    while (l <= r) {
        T m = midpoint(l, r);
        if (check(m))
            ret ? chmax(ret, m) : ret = m, l = m + 1;
        else
            r = m - 1;
    }
    return ret;
}
template <typename T>
optional<T> minCheck(T l, T r, auto check) {
    optional<T> ret;
    while (l <= r) {
        T m = midpoint(l, r);
        if (check(m))
            ret ? chmin(ret, m) : ret = m, r = m - 1;
        else
            l = m + 1;
    }
    return ret;
}
```

6.4 Merge Interals

Time: $(N \log N)$
Warning: It destroys the original array

```
#include "../Contest/template.cpp"
template <typename T>
vector<pair<T, T>> merge_intervals(vector<pair<T, T>> &intervals) {
    if (!len(intervals)) return {};
    using Pt = pair<T, T>;
    sort(all(intervals));
    vector<Pt> ret{intervals.front()};
    rep(i, 1, len(ret)) {
        auto &[pl, pr] = ret.back();
        auto &[l, r] = intervals[i];
        if (l <= pr)
            chmax(pr, r);
        else
            ret.eb(l, r);
    }
    return ret;
}
```

6.5 Mo's algorithm

```
template <typename T, typename Tans>
struct Mo {
    struct Query {
        int l, r, idx, block;
        Query(int _l, int _r, int _idx, int _block)
            : l(_l), r(_r), idx(_idx), block(_block) {}
        bool operator<(const Query &q) const {
            if (block != q.block) return block < q.block;
            return (block & 1 ? (r < q.r) : (r > q.r));
        }
    };
    vector<T> vs;
    vector<Query> qs;
    const int block_size;
    Mo(const vector<T> &a) : vs(a), block_size((int)ceil(sqrt(a.size())))
    {}
    void add_query(int l, int r) {
        qs.emplace_back(l, r, qs.size(), l / block_size);
    }
    auto solve() {
        // get answer return type
        vector<Tans> answers(qs.size());
        sort(all(qs));
        int cur_l = 0, cur_r = -1;
        for (auto q : qs) {
            while (cur_l > q.l) add(--cur_l);
            while (cur_r < q.r) add(++cur_r);
            while (cur_l < q.l) remove(cur_l++);
            while (cur_r > q.r) remove(cur_r--);
            answers[q.idx] = get_answer();
        }
        return answers;
    }
   private:
    // add value at idx from data structure
    inline void add(int idx) {}
    // remove value at idx from data structure
    inline void remove(int idx) {}
    // extract current answer of the data structure
    inline Tans get_answer() {}
};
```

6.6 __int128t stream

```
void print(__int128 x) {
    if (x < 0) {
        cout << '-';
        x = -x;
    }
    if (x > 9) print(x / 10);
    cout << (char)((x % 10) + '0');
}
__int128 read() {
    string s;
    cin >> s;
    __int128 x = 0;
    for (auto c : s) {
        if (c != '-') x += c - '0';
        x *= 10;
    }
    x /= 10;
    if (s[0] == '-') x = -x;
    return x;
}
```

7 Geometry

7.1 All i know about 2D stuff

Time: $O(N)$

```
#include <iterator>
#include "../Contest/template.cpp"
/*
    ====================================================================================
    */
const double EPS{1e-4};
const ld PI = acos(-1);
```

```cpp
enum PointPosition { IN, ON, OUT };
template <class Point>
vector<Point> segInter(Point a, Point b, Point c, Point d);
template <typename T>
bool equals(T a, T b) {
    if (std::is_floating_point<T>::value)
        return fabs(a - b) < EPS;
    else
        return a == b;
}
/*
    ============================================================================
    */
template <class T>
struct Point {
    typedef Point P;
    T x, y;
    explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
    bool operator<(P p) { return tie(x, y) < tie(p.x, p.y); }
    bool operator>(P& rhs) { return rhs < *this; }
    bool operator==(P p) { return tie(x, y) == tie(p.x, p.y); }
    P operator+(P p) { return P(x + p.x, y + p.y); }
    P operator-(P p) { return P(x - p.x, y - p.y); }
    P operator*(T d) { return P(x * d, y * d); }
    P operator/(T d) { return P(x / d, y / d); }
    T dot(P p) { return x * p.x + y * p.y; }
    T cross(P p) { return x * p.y - y * p.x; }
    T cross(P a, P b) { return (a - *this).cross(b - *this); }
    T dist2() { return x * x + y * y; }
    double dist() { return sqrt((double)dist2()); }
    // angle to x-axis in interval [-pi, pi]
    double angle() { return atan2(y, x); }
    P unit() { return *this / dist(); }  // makes dist()=1
    P perp() { return P(-y, x); }        // rotates +90 degrees
    P normal() { return perp().unit(); }
    // returns point rotated 'a' radians ccw around
    // the origin
    P rotate(double a) {
        return P(x * cos(a) - y * sin(a), x * sin(a) + y * cos(a));
    }
    pair<T, T> slope(Point<T>& o) {
        auto a = o.x - x;
        auto b = o.y - y;
        if (!is_floating_point<T>::value) {
            auto g = __gcd(a, b);
            if (g) a /= g, b /= g;
        }
        return {b, a};
    }
    friend ostream& operator<<(ostream& os, P p) {
        return os << "(" << p.x << "," << p.y << ")";
    }
    double distanceTo(Point<T>& other) {
        return hypot(other.x - x, other.y - y);
    }
};
/*
    ============================================================================
    */
template <typename T>
struct Line {
    T a, b, c;
    Point<T> p1, p2;
    Line(T a = 0, T b = 0, T c = 0) : a(a), b(b), c(c) {
        if (a != 0) {
            double x = 0;
            double y = (-c) / b;
            p1 = Point<T>(x, y);
        }
        if (b != 0) {
            double y = 0;
            double x = (-c) / a;
            p2 = Point<T>(x, y);
        }
    }
    Line(Point<T>& p, Point<T>& q) {
        a = p.y - q.y;
        b = q.x - p.x;
        c = p.cross(q);
        p1 = p, p2 = q;
    }
    bool operator==(Line<T>& other) {
        return tie(a, b, c) == tie(other.a, other.b, other.c);
    }
    // Less-than operator
    bool operator<(Line& rhs) {
        return tie(a, b, c) < tie(rhs.a, rhs.b, rhs.c);
    }
    bool operator>(Line& rhs) { return rhs < *this; }
    Line<T> norm() {
        T d = a == 0 ? b : a;
        return Line(a / d, b / d, c / d);
    }
    bool contains(Point<T>& p) { return equals(a * p.x + b * p.y + c, (T)
    0); }
    bool parallel(Line<T>& r) {
        auto det = a * r.b - b * r.a;
        return equals(det, 0) and !(*this == r);
    }
    bool orthogonal(Line<T>& r) { return equals(a * r.a + b * r.b, 0); }
    T direction(Point<T>& p3) { return p1.cross(p2, p3); }
    friend ostream& operator<<(ostream& os, Line l) {
        return os << fixed << setprecision(6) << "(" << l.a << "," << l.b
    << ","
                  << l.c << ")";
    }
    double distance(Point<T>& p) {
        return (a * p.x + b * p.y + c) / hypot(a, b);
    }
    Point<T> closest(Point<T>& p) {
        auto den = (a * a + b * b);
```

```cpp
        auto x = (b * (b * p.x - a * p.y) - a * c) / den;
        auto y = (a * (-b * p.x + a * p.y) - b * c) / den;
        return Point<T>{x, y};
    }
};
/*
    ================================================================================
    */
template <typename T>
struct LineSegment {
    Point<T> p1, p2;
    LineSegment(Point<T> p, Point<T> q) { p1 = p, p2 = q; }
    LineSegment(T a, T b, T c, T d)
        : LineSegment(Point<T>(a, b), Point<T>(c, d)) {}
    bool operator==(LineSegment<T>& other) {
        return tie(p1, p2) == tie(other.p1, other.p2);
    }
    // Less-than operator
    bool operator<(LineSegment& rhs) {
        return tie(p1, p2) < tie(rhs.p1, rhs.p2);
    }
    bool operator>(LineSegment& rhs) { return rhs < *this; }
    T direction(Point<T>& p3) { return p1.cross(p2, p3); }
    friend ostream& operator<<(ostream& os, LineSegment l) {
        return os << "(" << l.p1 << "," << l.p2 << ")";
    }
    vector<Point<T>> intersection(LineSegment<T>& other) {
        return segInter(p1, p2, other.p1, other.p2);
    }
    // Verifica se o ponto P da reta r que écontm A e B  pertence ao
    segmento
    bool contains(Point<T>& P) {
        return equals(p1.x, p2.x)
                   ? min(p1.y, p2.y) <= P.y and P.y <= max(p1.y, p2.y)
                   : min(p1.x, p2.x) <= P.x and P.x <= max(p1.x, p2.x);
    }
    // Ponto mais óprximo de P no segmento AB
    Point<T> closest(Point<T>& P) {
        Line<T> r(p1, p2);
        auto Q = r.closest(P);
        if (this->contains(Q)) return Q;
        auto distp1 = P.distanceTo(p1);
        auto distp2 = P.distanceTo(p2);
        if (distp1 <= distp2)
            return p1;
        else
            return p2;
    }
};
/*
    ================================================================================
    */
template <typename T>
struct Circle {
    Point<T> c;
    T r;
    Circle(Point<T> _c, T _r) : c(_c), r(_r) {}
    Circle(T _r) : Circle(Point<T>(0, 0), _r) {}
    ld area() const { return PI * r * r; }
    ld perimeter() const { return 2.0 * PI * r; }
    ld arc(ld theta) const { return theta * r; }
    ld chord(ld theta) const { return 2.0 * r * sin(theta / 2.0); }
    ld sector(ld theta) const { return (theta * r * r) / 2.0; }
    ld segment(ld theta) const { return ((theta - sin(theta)) * r * r) /
    2.0; }
    PointPosition position(const Point<T>& p) const {
        auto d = c.dist(p);
        return equals(d, r) ? ON : (d < r ? IN : OUT);
    }
};
/*
    ================================================================================
    */
template <typename T>
struct Rectangle {
    Point<T> P, Q;
    T b, h;
    Rectangle(const Point<T>& p, const Point<T>& q) : P(P), Q(q) {
        assert(P != Q);
        b = max(P.x, Q.x) - min(P.x, Q.x);
        h = max(P.y, Q.y) - min(P.y, Q.y);
    }
    Rectangle(T base, T height)
        : P(0, 0), Q(base, height), b(base), h(height) {}
    T perimeter() const { return 2 * b + 2 * h; }
    T area() const { return b * h; }
    optional<Rectangle> intersection(const Rectangle& r) const {
        using pt = pair<T, T>;
        auto i = pt(min(P.x, Q.x), max(P.x, Q.x));
        auto u = pt(min(r.P.x, r.Q.x), max(r.P.x, r.Q.x));
        auto a = max(i.first, u.first);
        auto b = min(i.second, u.second);
        i = pt(min(P.y, Q.y), max(P.y, Q.y));
        u = pt(min(r.P.y, r.Q.y), max(r.P.y, r.Q.y));
        auto c = max(i.first, u.first);
        auto d = max(i.second, u.first);
        if (d < c or b < a) return nullopt;
        return {{a, c}, {b, d}};
    }
};
/*
    ================================================================================
    */
template <typename T>
struct Trapezium {
    T B, b, h;
    T area() const { return ((b + B) * h) / 2; }
};
/*
```

```
    ================================================================================
    */
template <typename T>
struct Triangle {
    Point<T> A, B, C;
    enum SidesClass { EQUILATERAL, ISOCELES, SCALENE };
    SidesClass classification_by_sides() const {
        auto a = A.distanceTo(B);
        auto b = B.distanceTo(C);
        auto c = C.distanceTo(A);
        if (equals(a, b) && equals(b, c)) return EQUILATERAL;
        if (equals(a, b) or equals(a, c) or equals(b, c)) return ISOCELES;
        return SCALENE;
    }
    enum AnglesClass { RIGHT, ACUTE, OBTUSE };
    AnglesClass classification_by_angles() const {
        auto a = dist(A, B);
        auto b = dist(B, C);
        auto c = dist(C, A);
        auto alpha = acos((a * a - b * b - c * c) / (-2 * b * c));
        auto beta = acos((b * b - a * a - c * c) / (-2 * a * c));
        auto gamma = acos((c * c - a * a - b * b) / (-2 * a * b));
        auto right = PI / 2.0;
        if (equals(alpha, right) || equals(beta, right) || equals(gamma,
    right))
            return RIGHT;
        if (alpha > right || beta > right || gamma > right) return OBTUSE;
        return ACUTE;
    }
    double perimeter() const {
        auto a = dist(A, B), b = dist(B, C), c = dist(C, A);
        return a + b + c;
    }
    double area() const {
        Line<T> r(A, B);
        auto b = dist(A, B);
        auto h = r.distance(C);
        return (b * h) / 2;
    }
};
template <typename T>
Point<T> triangleBarycenter(const Point<T>& a, const Point<T>& b,
                            const Point<T>& c) {
    return Point<T>((a.x + b.x + c.x) / 3.0, (a.y + b.y + c.y) / 3.0);
}
template <typename T>
Point<T> triangleOrthocenter(const Point<T>& a, const Point<T>& b,
                             const Point<T>& c) {
    Line<T> r(a, b), s(a, c);
    Line<T> u{r.b, -r.a, -(c.x * r.b - c.y * r.a)};
    Line<T> v{s.b, -s.a, -(b.x * s.b - b.y * s.a)};
    auto det = u.a * v.b - u.b * v.a;
    auto x = (-u.c * v.b + v.c * u.b) / det;
    auto y = (-v.c * u.a + u.c * v.a) / det;
    return {x, y};
}
template <typename T>
Point<double> triangleIncenter(const Point<T>& a, const Point<T>& b,
                               const Point<T>& c) {
    auto dab = distance(a, b);
    auto dbc = distance(b, c);
    auto dca = distance(c, a);
    auto p = dab + dbc + dca;
    auto x = (a.x * dab + b.x * dbc + b.x * dca) / (p);
    auto y = (a.y * dab + b.y * dbc + b.y * dca) / (p);
    return Point<double>(x, y);
}
template <typename T>
Point<T> triangleCircumcenter(const Point<T>& A, const Point<T>& B,
                              const Point<T>& C) {
    auto D = 2 * (A.x * (B.y - C.y) + B.x * (C.y - A.y) + C.x * (A.y - B.y
    ));
    auto A2 = A.x * A.x + A.y * A.y;
    auto B2 = B.x * B.x + B.y * B.y;
    auto C2 = C.x * C.x + C.y * C.y;
    auto x = (A2 * (B.y - C.y) + B2 * (C.y - A.y) + C2 * (A.y - B.y)) / D;
    auto y = (A2 * (C.x - B.x) + B2 * (A.x - C.x) + C2 * (B.x - A.x)) / D;
    return {x, y};
}
template <typename T>
Point<T> triangleCircumradius(const Point<T>& a, const Point<T>& b,
                              const Point<T>& c) {
    auto dab = distance(a, b);
    auto dbc = distance(b, c);
    auto dca = distance(c, a);
    return (dab + dbc + dca) / triangleArea(a, b, c);
}
/*
    ================================================================================
    */
template <class Point>
vector<Point> segInter(Point a, Point b, Point c, Point d) {
    auto oa = c.cross(d, a), ob = c.cross(d, b), oc = a.cross(b, c),
         od = a.cross(b, d);
    // Checks if intersection is single non-endpoint
    // point.
    if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
        return {(a * ob - b * oa) / (ob - oa)};
    set<Point> s;
    if (onSegment(c, d, a)) s.insert(a);
    if (onSegment(c, d, b)) s.insert(b);
    if (onSegment(a, b, c)) s.insert(c);
    if (onSegment(a, b, d)) s.insert(d);
    return {all(s)};
}
/*
    ================================================================================
```

```
    */
template <typename T>
double angle(const Point<T>& P, const Point<T>& Q, const Point<T>& R,
             const Point<T>& S) {
    auto ux = P.x - Q.x;
    auto uy = P.y - Q.y;
    auto vx = R.x - S.x;
    auto vy = R.y - S.y;
    auto num = ux * vx + uy * vy;
    auto den = hypot(ux, uy) * hypot(vx, vy);
    // Caso especial: se den == 0, algum dos vetores é degenerado: os dois
    // pontos ãso iguais. Neste caso, o ângulo ãno áest definido
    return acos(num / den);
}
/*
    ===========================================================================
    */
struct pt {
    double x, y;
    int id;
};
int orientation(pt a, pt b, pt c) {
    double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) + c.x * (a.y - b.y);
    if (v < 0) return -1;  // clockwise
    if (v > 0) return +1;  // counter-clockwise
    return 0;
}
bool cw(pt a, pt b, pt c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }
void convex_hull(vector<pt>& pts, bool include_collinear = false) {
    pt p0 = *min_element(all(pts), [](pt a, pt b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(all(pts), [&p0](const pt& a, const pt& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x - a.x) * (p0.x - a.x) + (p0.y - a.y) * (p0.y - a.
    y) <
                   (p0.x - b.x) * (p0.x - b.x) + (p0.y - b.y) * (p0.y - b.
    y);
        return o < 0;
    });
    if (include_collinear) {
        int i = len(pts) - 1;
        while (i >= 0 && collinear(p0, pts[i], pts.back())) i--;
        reverse(pts.begin() + i + 1, pts.end());
    }
    vector<pt> st;
    for (int i = 0; i < len(pts); i++) {
        while (st.size() > 1 &&
               !cw(st[len(st) - 2], st.back(), pts[i], include_collinear))
            st.pop_back();
        st.push_back(pts[i]);
    }
    pts = st;
}
/*
    ===========================================================================
    */
template <typename T>
double ccRadius(const Point<T>& A, const Point<T>& B, const Point<T>& C) {
    return (B - A).dist() * (C - B).dist() * (A - C).dist() /
           abs((B - A).cross(C - A)) / 2;
}
template <typename T>
Point<T> ccCenter(const Point<T>& A, const Point<T>& B, const Point<T>& C)
 {
    Point<T> b = C - A, c = B - A;
    return A + (b * c.dist2() - c * b.dist2()).perp() / b.cross(c) / 2;
}
template <typename T>
pair<Point<T>, double> mec(vector<Point<T>> ps) {
    shuffle(all(ps), mt19937(time(0)));
    Point<T> o = ps[0];
    double r = 0, EPS = 1 + 1e-8;
    rep(i, 0, len(ps)) if ((o - ps[i]).dist() > r * EPS) {
        o = ps[i], r = 0;
        rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
            o = (ps[i] + ps[j]) / 2;
            r = (o - ps[i]).dist();
            rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
                o = ccCenter(ps[i], ps[j], ps[k]);
                r = (o - ps[i]).dist();
            }
        }
    }
    return {o, r};
}
/*
    ===========================================================================
    */
template <typename T>
Line<T> perpendicular_bisector(const Point<T>& P, const Point<T>& Q) {
    auto a = 2 * (Q.x - P.x);
    auto b = 2 * (Q.y - P.y);
    auto c = (P.x * P.x + P.y * P.y) - (Q.x * Q.x + Q.y * Q.y);
    return {a, b, c};
}
/*
    ===========================================================================
    */
ll cross(ll x1, ll y1, ll x2, ll y2) { return x1 * y2 - x2 * y1; }
ll polygonArea(vector<pll>& pts) {
    ll ats = 0;
```

```
    for (int i = 2; i < len(pts); i++)
        ats += cross(pts[i].first - pts[0].first, pts[i].second - pts[0].
    second,
                     pts[i - 1].first - pts[0].first,
                     pts[i - 1].second - pts[0].second);
    return abs(ats / 2ll);
}
ll boundary(vector<pll>& pts) {
    ll ats = pts.size();
    for (int i = 0; i < len(pts); i++) {
        ll deltax = (pts[i].first - pts[(i + 1) % pts.size()].first);
        ll deltay = (pts[i].second - pts[(i + 1) % pts.size()].second);
        ats += abs(__gcd(deltax, deltay)) - 1;
    }
    return ats;
}
pll latticePoints(vector<pll>& pts) {
    ll bounds = boundary(pts);
    ll area = polygonArea(pts);
    ll inside = area + 1ll - bounds / 2ll;
    return {inside, bounds};
}
/*
    ============================================================================
    */
template <typename T>
bool contains(const Point<T>& A, const Point<T>& B, const Point<T>& P) {
    // Verifica se P áest na ãregio retangular
    auto xmin = min(A.x, B.x);
    auto xmax = max(A.x, B.x);
    auto ymin = min(A.y, B.y);
    auto ymax = max(A.y, B.y);
    if (P.x < xmin || P.x > xmax || P.y < ymin || P.y > ymax) return false
    ;
    // Verifica çãrelao de çsemelhana no âtringulo
    return equals((P.y - A.y) * (B.x - A.x), (P.x - A.x) * (B.y - A.y));
}
/*
    ============================================================================
    */
// the polygon area of a intersection between a circle and a ccw polygon
template <typename T>
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(Point<T> c, double r, vector<Point<T>> ps) {
    auto tri = [&](Point<T> p, Point<T> q) {
        auto r2 = r * r / 2;
        Point<T> d = q - p;
        auto a = d.dot(p) / d.dist2(), b = (p.dist2() - r * r) / d.dist2()
    ;
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a - sqrt(det)), t = min(1., -a + sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        Point<T> u = p + d * s, v = p + d * t;
        return arg(p, u) * r2 + u.cross(v) / 2 + arg(v, q) * r2;
    };
    auto sum = 0.0;
    rep(i, 0, len(ps)) sum += tri(ps[i] - c, ps[(i + 1) % len(ps)] - c);
    return sum;
}
/*
    ============================================================================
    */
bool checkIfPolygonIsConvex(vector < Point<T>) {
    if (n < 3) return false;
}
```

7.2 Angle between three points

Description: Computes the angle apb in radians
Warning: a is equal to b then the angle isn't defined.

```
#include "./template.cpp"
template <typename T>
ld angle(const Point<T>& p, const Point<T>& a, const Point<T>& b) {
    auto ux = p.x - a.x;
    auto uy = p.y - a.y;
    auto vx = p.x - b.x;
    auto vy = p.y - b.y;
    auto num = ux * vx + uy * vy;
    auto den = hypot(ux, uy) * hypot(vx, vy);
    return acos(num / den);
}
```

7.3 Area of union of rectangles

```
using SegT = ll;
const SegT eSeg = 1e9;
struct QueryT {
    SegT q, v;
    QueryT() : q(0), v(eSeg) {}
    QueryT(SegT _v) : q(1), v(_v) {}
};
inline QueryT combine(QueryT ln, QueryT rn, pii lr1, pii lr2) {
    QueryT ret;
    if (ln.v < rn.v) ret = ln;
    if (rn.v < ln.v) ret = rn;
    if (rn.v == ln.v) {
        ret.v = ln.v;
        ret.q = ln.q + rn.q;
    }
    return ret;
}
using LazyT = SegT;
inline QueryT applyLazyInQuery(QueryT q, LazyT l, pii lr) {
    if (l == LazyT()) return q;
    if (q.v == eSeg) q.v = 0, q.q = 1;
```

```cpp
    q.v += l;
    return q;
}
inline LazyT applyLazyInLazy(LazyT a, LazyT b) { return a + b; }
using UpdateT = SegT;
inline QueryT applyUpdateInQuery(QueryT q, UpdateT u, pii lr) {
    return applyLazyInQuery(q, u, lr);
}
inline LazyT applyUpdateInLazy(LazyT l, UpdateT u, pii lr) { return l + u;
     }
template <typename Qt = QueryT, typename Lt = LazyT, typename Ut = UpdateT
    ,
          auto C = combine, auto ALQ = applyLazyInQuery,
          auto ALL = applyLazyInLazy, auto AUQ = applyUpdateInQuery,
          auto AUL = applyUpdateInLazy>
struct LazySegmentTree {
    int n, h;
    vector<Qt> ts;
    vector<Lt> ds;
    vector<pii> lrs;
    LazySegmentTree(int _n)
        : n(_n),
          h(sizeof(int) * 8 - __builtin_clz(n)),
          ts(n << 1),
          ds(n),
          lrs(n << 1) {
        rep(i, 0, n) lrs[i + n] = {i, i};
        rrep(i, n - 1, 0) {
            lrs[i] = {lrs[i << 1].first, lrs[i << 1 | 1].second};
        }
    }
    LazySegmentTree(const vector<Qt> &xs) : LazySegmentTree(len(xs)) {
        copy(all(xs), ts.begin() + n);
        rep(i, 0, n) lrs[i + n] = {i, i};
        rrep(i, n - 1, 0) {
            ts[i] = C(ts[i << 1], ts[i << 1 | 1], lrs[i << 1], lrs[i << 1
    | 1]);
        }
    }
    void set(int p, Qt v) {
        ts[p + n] = v;
        build(p + n);
    }
    void upd(int l, int r, Ut v) {
        l += n, r += n + 1;
        int l0 = l, r0 = r;
        for (; l < r; l >>= 1, r >>= 1) {
            if (l & 1) apply(l++, v);
            if (r & 1) apply(--r, v);
        }
        build(l0), build(r0 - 1);
    }
    Qt qry(int l, int r) {
        l += n, r += n + 1;
        push(l), push(r - 1);
        Qt resl = Qt(), resr = Qt();
        pii lr1 = {l, l}, lr2 = {r, r};
        for (; l < r; l >>= 1, r >>= 1) {
            if (l & 1) resl = C(resl, ts[l], lr1, lrs[l]), l++;
            if (r & 1) r--, resr = C(ts[r], resr, lrs[r], lr2);
        }
        return C(resl, resr, lr1, lr2);
    }
    void build(int p) {
        while (p > 1) {
            p >>= 1;
            ts[p] =
                ALQ(C(ts[p << 1], ts[p << 1 | 1], lrs[p << 1], lrs[p << 1
    | 1]),
                    ds[p], lrs[p]);
        }
    }
    void push(int p) {
        rrep(s, h, 0) {
            int i = p >> s;
            if (ds[i] != Lt()) {
                apply(i << 1, ds[i]), apply(i << 1 | 1, ds[i]);
                ds[i] = Lt();
            }
        }
    }
    inline void apply(int p, Ut v) {
        ts[p] = AUQ(ts[p], v, lrs[p]);
        if (p < n) ds[p] = AUL(ds[p], v, lrs[p]);
    }
};
ll areaOfRectanglesUnion(
    const vector<pair<Point<int>, Point<int>>> &rectangles) {
    if (!size(rectangles)) return 0;
    int maxy = INT_MIN;
    for (auto &[p1, p2] : rectangles) {
        assert(p1.x < p2.x && p1.y < p2.y);
        maxy = max({maxy, p1.y, p2.y});
    }
    vector<array<int, 4>> sl;
    sl.reserve(size(rectangles) * 2);
    for (auto &[p1, p2] : rectangles) {
        sl.push_back({p1.x, p1.y, p2.y - 1, 1});
        sl.push_back({p2.x, p1.y, p2.y - 1, -1});
    }
    sort(sl.begin(), sl.end());
    vector<QueryT> aux(maxy, QueryT(0));
    LazySegmentTree seg(aux);
    // memset(seg_vec, 0, sizeof(ll) * maxy);
    // seg::build(maxy, seg_vec);
    int prevx = get<0>(sl.front());
    ll ans = 0;
```

```
    for (auto &[curx, ys, yf, inc] : sl) {
        auto [q, v] = seg.qry(0, maxy - 1);
        // auto [q, v] = seg::query(0, maxy - 1);
        ans += (ll)(curx - prevx) * (v ? maxy : maxy - q);
        seg.upd(ys, yf, inc);
        prevx = curx;
    }
    return ans;
}
```

7.4 Area: polygon

```
#include "./template.cpp"
template <typename T>
ld area(const vector<Point<T>>& pts) {
    ld a = 0.0;
    int n = size(pts);
    for (int i = 0; i < n; i++) {
        a += pts[i].x * pts[(i + 1) % n].y;
        a -= pts[i].y * pts[(i + 1) % n].x;
    }
    return fabs(a) / (ld)2;
}
```

7.5 Check if point belongs to line

```
#pragma once
#include "./Define line from two points.cpp"
#include "./template.cpp"
template <typename T>
bool lineContainsPoint(const Point<T>& r, const Point<T>& p,
                       const Point<T>& q) {
    auto [a, b, c] = defineLine(p, q);
    return equals((T)0, a * r.x + b * r.y + c);
}
```

7.6 Check if point belongs to segment

```
#include "./template.cpp"
template <class P>
bool segmentContainsPoint(const P& p, const P& a, const P& b) {
    auto xmin = min(a.x, b.x);
    auto xmax = max(a.x, b.x);
    auto ymin = min(a.y, b.y);
    auto ymax = max(a.y, b.y);
    if (p.x < xmin or p.x > xmax or p.y < ymin or p.y > ymax) return false
    ;
    return equals((p.y - a.y) * (b.x - a.x), (p.x - a.x) * (b.y - a.y));
}
```

7.7 Check if point is inside polygon

Description: checks if the point p is inside the polygon with vertices in pts, works for both convex and concave polygons.

```
#pragma once
#include "./Angle between three points.cpp"
#include "./Check if point belongs to segment.cpp"
#include "./Determinant.cpp"
#include "./template.cpp"
template <typename T>
bool contains(const vector<Point<T>>& pts, const Point<T>& p) {
    int n = size(pts);
    if (n < 3) return false;  // may treat it appart
    T sum = 0.0;
    for (int i = 0; i < n; i++) {
        auto d = determinant(p, pts[i], pts[(i + 1) % n]);
        auto a = angle(p, pts[i], pts[(i + 1) % n]);
        sum += d > 0 ? a : (d < 0 ? -a : 0);
    }
    return equals(fabs(sum), 2 * PI);
}
// 0: outside, 1: inside, 2: boundary
template <class P>
int pointInPolygon(const vector<P>& pts, const P& p) {
    if (contains(pts, p)) return 1;
    int n = size(pts);
    for (int i = 0; i < n; i++) {
        if (segmentContainsPoint(p, pts[i], pts[(i + 1) % n])) {
            return 2;
        }
    }
    return 0;
}
```

7.8 Convex hull

```
#include "../Contest/template.cpp"
#include "./Determinant.cpp"
#include "./template.cpp"
template <typename T>
vector<Point<T>> convexHull(vector<Point<T>> pts) {
    if (len(pts) <= 1) return pts;
    sort(all(pts));
    vector<Point<T>> h(len(pts) + 1);
    int s = 0, t = 0;
    for (int it = 2; it--; s = --t, reverse(all(pts)))
        for (Point<T> p : pts) {
            while (t >= s + 2 && determinant(h[t - 2], h[t - 1], p) <= 0)
    t--;
            h[t++] = p;
        }
```

```
    return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
template <typename T>
vector<Point<T>> convexHull2(vector<Point<T>> pts) {
    int n = len(pts);
    sort(pts.begin(), pts.end());
    vector<Point<T>> l, u;
    for (int i = 0; i < n; i++) {
        while (len(l) >= 2 &&
               determinant(l[len(l) - 1], l[len(l) - 2], pts[i]) < 0) {
            l.pop_back();
        }
        l.push_back(pts[i]);
    }
    for (int i = n - 1; ~i; --i) {
        while (len(u) >= 2 &&
               determinant(u[len(u) - 1], u[len(u) - 2], pts[i]) < 0) {
            u.pop_back();
        }
        u.push_back(pts[i]);
    }
    u.pop_back(), l.pop_back();
    u.reserve(len(u) + len(l));
    u.insert(u.end(), all(l));
    return u;
}
```

7.9 Cross product between points

```
#pragma once
#include "./template.cpp"
template <typename T>
T cross(const Point<T>& p, const Point<T>& q) {
    return p.x * q.y - p.y * q.x;
}
```

7.10 Define line from two points

```
#pragma once
#include "./template.cpp"
template <typename T>
inline tuple<T, T, T> defineLine(const Point<T>& p, const Point<T>& q) {
    return {p.y - q.y, q.x - p.x, cross(p, q)};
}
```

7.11 Determinant

```
#pragma once
#include "./template.cpp"
template <typename T>
T determinant(const Point<T>& p, const Point<T>& q, const Point<T>& r) {
    return (p.x * q.y + p.y * r.x + q.x * r.y) -
           (r.x * q.y + r.y * p.x + q.x * p.y);
}
```

7.12 Distance: point to point

```
#include "./template.cpp"
template <typename T>
T distance(const Point<T>& p, const Point<T>& q) {
    return hypot(p.x - q.x, p.y - q.y);
}
```

7.13 Halfplane intersection

```
#pragma once
#include "./Point.cpp"
#include "./template.cpp"
// Basic half-plane struct.
struct Halfplane {
    // 'p' is a passing point of the line and 'pq' is the direction vector
     of
    // the line.
    Point<ld> p, pq;
    long double angle;
    Halfplane() {}
    Halfplane(const Point<ld>& a, const Point<ld>& b) : p(a), pq(b - a) {
        angle = atan2l(pq.y, pq.x);
    }
    // Check if point 'r' is outside this half-plane.
    // Every half-plane allows the region to the LEFT of its line.
    bool out(const Point<ld>& r) { return cross(pq, r - p) < -EPS; }
    // Intersection point of the lines of two half-planes. It is assumed
    they're
    // never parallel.
    friend Point<ld> inter(const Halfplane& s, const Halfplane& t) {
        long double alpha = cross((t.p - s.p), t.pq) / cross(s.pq, t.pq);
        return s.p + (s.pq * alpha);
    }
};
// Actual algorithm
// receive it by reference if don't care messing with it
vector<Point<ld>> hp_intersect(vector<Halfplane> H) {
    const ld inf = 2e6;
    Point<ld> box[4] = {// Bounding box in CCW order
                        Point<ld>(inf, inf), Point<ld>(-inf, inf),
                        Point<ld>(-inf, -inf), Point<ld>(inf, -inf)};
    for (int i = 0; i < 4; i++) {  // Add bounding box half-planes.
        Halfplane aux(box[i], box[(i + 1) % 4]);
```

```
    H.push_back(aux);
}
// Sort by angle and start algorithm
sort(H.begin(), H.end(), [&](const Halfplane& a, const Halfplane& b) {
    return a.angle < b.angle;
});
deque<Halfplane> dq;
int len = 0;
for (int i = 0; i < int(H.size()); i++) {
    // Remove from the back of the deque while last half-plane is
redundant
    while (len > 1 && H[i].out(inter(dq[len - 1], dq[len - 2]))) {
        dq.pop_back();
        --len;
    }
    // Remove from the front of the deque while first half-plane is
    // redundant
    while (len > 1 && H[i].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }
    // Special case check: Parallel half-planes
    if (len > 0 && fabsl(cross(H[i].pq, dq[len - 1].pq)) < EPS) {
        // Opposite parallel half-planes that ended up checked against
 each
        // other.
        if (dot(H[i].pq, dq[len - 1].pq) < 0.0) return vector<Point<ld
>>();
        // Same direction half-plane: keep only the leftmost half-
plane.
        if (H[i].out(dq[len - 1].p)) {
            dq.pop_back();
            --len;
        } else
            continue;
    }
    // Add new half-plane
    dq.push_back(H[i]);
    ++len;
}
// Final cleanup: Check half-planes at the front against the back and
// vice-versa
while (len > 2 && dq[0].out(inter(dq[len - 1], dq[len - 2]))) {
    dq.pop_back();
    --len;
}
while (len > 2 && dq[len - 1].out(inter(dq[0], dq[1]))) {
    dq.pop_front();
    --len;
}
// Report empty intersection if necessary
if (len < 3) return vector<Point<ld>>();
// Reconstruct the convex polygon from the remaining half-planes.
    vector<Point<ld>> ret(len);
    for (int i = 0; i + 1 < len; i++) {
        ret[i] = inter(dq[i], dq[i + 1]);
    }
    ret.back() = inter(dq[len - 1], dq[0]);
    return ret;
}
```

7.14 Lattice points

```
#pragma once
#include "../Contest/template.cpp"
#include "./Area: polygon.cpp"
#include "./template.cpp"
template <typename T>
pair<ll, ll> latticePoints(const vector<Point<T>> &pts) {
    ll bounds = pts.size();
    int n = pts.size();
    for (int i = 0; i < n; i++) {
        ll deltax = (pts[i].x - pts[(i + 1) % n].x);
        ll deltay = (pts[i].y - pts[(i + 1) % n].y);
        bounds += abs(__gcd(deltax, deltay)) - 1;
    }
    ll a = area(pts);
    ll inside = a + 1 - bounds / 2ll;
    return {inside, bounds};
}
```

7.15 Left of polygon cut

Warning: if some vertex lies exactly on the line A B, theese vertex will be included in teh answer

```
#include "./Determinant.cpp"
#include "./template.cpp"
template <typename T>
vector<Point<T>> leftOfPolygonCut(const vector<Point<T>>& vs, const Point<
    T>& A,
                                  const Point<T>& B) {
    // çãInterseo entre a reta AB e o segmento de reta PQ
    auto intersection = [&](const Point<T>& P, const Point<T>& Q,
                            const Point<T>& A, const Point<T>& B) -> Point
    <T> {
        auto a = B.y - A.y;
        auto b = A.x - B.x;
        auto c = B.x * A.y - A.x * B.y;
        auto u = fabs(a * P.x + b * P.y + c);
        auto v = fabs(a * Q.x + b * Q.y + c);
        // éMdia ponderada pelas âdistncias de P e Q éat a reta AB
        return {(P.x * v + Q.x * u) / (u + v), (P.y * v + Q.y * u) / (u +
    v)};
    };
    vector<Point<T>> points;
```

```
    int n = size(vs);
    for (int i = 0; i < n; ++i) {
        auto d1 = determinant(A, B, vs[i]);
        auto d2 = determinant(A, B, vs[(i + 1) % n]);
        // éVrtice à esquerda da reta
        if (d1 > -EPS) points.push_back(vs[i]);
        // A aresta cruza a reta
        if (d1 * d2 < -EPS)
            points.push_back(intersection(vs[i], vs[(i + 1) % n], A, B));
    }
    return points;
}
```

7.16 Perimeter: polygon

```
#include "./Distance: point to point.cpp"
#include "./template.cpp"
template <typename T>
T perimeter(const vector<Point<T>>& pts) {
    T p = 0.0;
    int n = size(pts);
    for (int i = 0; i < n; i++) {
        p += distance(pts[i], pts[(i + 1) % n]);
    }
    return p;
}
```

7.17 Point

```
// Basic point/vector struct.
template <typename T>
struct Point {
    T x, y;
    Point(T x = 0, T y = 0) : x(x), y(y) {}
    // Addition, substraction, multiply by constant, dot product, cross
    product.
    friend Point<T> operator+(const Point<T>& p, const Point<T>& q) {
        return Point<T>(p.x + q.x, p.y + q.y);
    }
    friend Point<T> operator-(const Point<T>& p, const Point<T>& q) {
        return Point<T>(p.x - q.x, p.y - q.y);
    }
    template <typename T2>
    friend Point<T> operator*(const Point<T>& p, T2 k) {
        return Point<T>(p.x * k, p.y * k);
    }
    friend T dot(const Point<T>& p, const Point<T>& q) {
        return p.x * q.x + p.y * q.y;
    }
    friend T cross(const Point<T>& p, const Point<T>& q) {
        return p.x * q.y - p.y * q.x;
    }
};
```

7.18 Polygon (regular): apothem

```
#include "./Distance: point to point.cpp"
#include "./template.cpp"
template <typename T>
ld apothem(const vector<Point<T>>& pts) {
    auto s = distance(pts[0], pts[1]);
    int n = size(pts);
    return (s / 2.0) * (1.0 / tan(PI / n));
}
```

7.19 Polygon (regular): circumradius

```
#include "./Distance: point to point.cpp"
#include "./template.cpp"
template <typename T>
ld circumradius(const vector<Point<T>>& pts) {
    auto s = distance(pts[0], pts[1]);
    int n = size(pts);
    return (s / 2.0) * (1.0 / sin(PI / (ld)n));
}
```

7.20 Polygon: check if is convex

```
#include "./Determinant.cpp"
#include "./template.cpp"
template <typename T>
bool checkIfPolygonIsConvex(vector<Point<T>>& pts) {
    int n = size(pts);
    if (n < 3) return false;
    int l, g, e;
    l = g = e = 0;
    for (int i = 0; i < n; i++) {
        auto d = determinant(pts[i], pts[(i + 1) % n], pts[(i + 2) % n]);
        d ? (d > 0 ? g++ : l++) : e++;
    }
    return l == n or g == n;
}
```

7.21 Rectangle intersection

```cpp
/*
   Assumes that the points P, Q that define
   a rectangle are the bottom-left and top-right
   corner, and also that the sides are parallel to the axis.
 */
#pragma once
#include "../Contest/template.cpp"
#include "./Point.cpp"
template <typename T>
optional<pair<Point<T>, Point<T>>> rectangleIntersection(
    const pair<Point<T>, Point<T>> &r1, const pair<Point<T>, Point<T>> &r2
    ) {
    assert(r1.first.x < r1.second.x && r1.first.y < r1.second.y);
    assert(r2.first.x < r2.second.x && r2.first.y < r2.second.y);
    T x1 = max(r1.first.x, r2.first.x);
    T x2 = min(r1.second.x, r2.second.x);
    T y1 = max(r1.first.y, r2.first.y);
    T y2 = min(r1.second.y, r2.second.y);
    if (x1 >= x2 or y1 >= y2) return nullopt;
    return pair<Point<T>, Point<T>>{{x1, y1}, {x2, y2}};
}
```

7.22 template

```cpp
#pragma once
#include <bits/stdc++.h>
using namespace std;
using ld = long double;
template <typename T>
using Point = pair<T, T>;
#define x first
#define y second
const double EPS{1e-6};
const ld PI = acos(-1);
template <typename T>
bool equals(T a, T b) {
    if (std::is_floating_point<T>::value)
        return fabs(a - b) < EPS;
    else
        return a == b;
}
template <typename T>
bool equals(Point<T> a, Point<T> b) {
    if (std::is_floating_point<T>::value)
        return fabs(a.x - b.x) < EPS && fabs(a.y - b.y) < EPS;
    else
        return a == b;
}
```

8 Graphs

8.1 Heavy-Light Decomposition (point update)

8.1.1 Maximum number on path

```cpp
struct Node {
    ll value;
    Node()
        : value(numeric_limits<ll>::min()) {};  // Neutral
                                                // element
    Node(ll v) : value(v) {};
};
Node combine(Node l, Node r) {
    Node m;
    m.value = max(l.value, r.value);
    return m;
}
template <typename T = Node, auto F = combine>
struct SegTree {
    int n;
    vector<T> st;
    SegTree(int _n) : n(_n), st(n << 1) {}
    void set(int p, const T &k) {
        for (st[p += n] = k; p >>= 1;) st[p] = F(st[p << 1], st[p << 1 |
    1]);
    }
    T query(int l, int r) {
        T ansl, ansr;
        for (l += n, r += n + 1; l < r; l >>= 1, r >>= 1) {
            if (l & 1) ansl = F(ansl, st[l++]);
            if (r & 1) ansr = F(st[--r], ansr);
        }
        return F(ansl, ansr);
    }
};
template <typename SegT = Node, auto SegOp = combine>
struct HeavyLightDecomposition {
    int n;
    vi ps, ds, sz, heavy, head, pos;
    SegTree<SegT, SegOp> seg;
    HeavyLightDecomposition(const vi2d &g, const vector<SegT> &v, int root
     = 0)
        : n(len(g)), seg(n) {
        ps = ds = sz = heavy = head = pos = vi(n, -1);
        auto dfs = [&](auto &&self, int u) -> void {
            sz[u] = 1;
            int mx = 0;
            for (auto x : g[u])
                if (x != ps[u]) {
                    ps[x] = u;
                    ds[x] = ds[u] + 1;
                    self(self, x);
                    sz[u] += sz[x];
```

```
                if (sz[x] > mx) mx = sz[x], heavy[u] = x;
            }
        };
        dfs(dfs, root);
        for (int i = 0, cur = 0; i < n; i++) {
            if (ps[i] == -1 or heavy[ps[i]] != i)
                for (int j = i; j != -1; j = heavy[j]) {
                    head[j] = i;
                    pos[j] = cur++;
                }
        }
        rep(i, 0, n) seg.set(pos[i], v[i]);
    }
    vector<pii> disjoint_ranges(int u, int v) {
        vector<pii> ret;
        for (; head[u] != head[v]; v = ps[head[v]]) {
            if (ds[head[u]] > ds[head[v]]) swap(u, v);
            ret.eb(pos[head[v]], pos[v]);
        }
        if (ds[u] > ds[v]) swap(u, v);
        ret.eb(pos[u], pos[v]);
        return ret;
    }
    SegT query_path(int u, int v) {
        SegT res;
        for (auto [l, r] : disjoint_ranges(u, v)) {
            res = SegOp(res, seg.query(l, r));
        }
        return res;
    }
    SegT query_subtree(int u) const {
        return seg.query(pos[u], pos[u] + sz[u] - 1);
    }
    void set(int u, SegT x) { seg.set(pos[u], x); }
};
```

8.2 2-SAT

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ becomes true, or reports that it is unsatisfiable.
Usage: Negated variables are represented by bit-inversions $(\tilde{x})$.
Returns true iff it is solvable ts.values[0..N-1] holds the assigned values to the vars.
Time: $O(N + E)$, where N is the number of boolean variables, and E is the number of clauses.

```
/
struct TwoSat {
    int N;
    vector<vi> gr;
    vi values;  // 0 = false, 1 = true
    TwoSat(int n = 0) : N(n), gr(2 * n) {}
    int addVar() {  // (optional)
        gr.eb();
        gr.eb();
        return N++;
    }
    void either(int f, int j) {
        f = max(2 * f, -1 - 2 * f);
        j = max(2 * j, -1 - 2 * j);
        gr[f].pb(j ^ 1);
        gr[j].pb(f ^ 1);
    }
    void setValue(int x) { either(x, x); }
    void implies(int f, int j) { either(~f, j); }  // (optional)
    void atMostOne(const vi &li) {  // (optional)
        if (len(li) <= 1) return;
        int cur = ~li[0];
        rep(i, 2, len(li)) {
            int next = addVar();
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }
    vi val, comp, z;
    int time = 0;
    int dfs(int i) {
        int low = val[i] = ++time, x;
        z.pb(i);
        for (int e : gr[i])
            if (!comp[e]) low = min(low, val[e] ?: dfs(e));
        if (low == val[i]) do {
                x = z.back();
                z.ppb();
                comp[x] = low;
                if (values[x >> 1] == -1) values[x >> 1] = x & 1;
            } while (x != i);
        return val[i] = low;
    }
    bool solve() {
        values.assign(N, -1);
        val.assign(2 * N, 0);
        comp = val;
        rep(i, 0, 2 * N) if (!comp[i]) dfs(i);
        rep(i, 0, N) if (comp[2 * i] == comp[2 * i + 1]) return 0;
        return 1;
    }
};
```

8.3 BFS-01

Description: Similar to a Dijkstra given a weighted graph finds the distance from source s to every other node.

Time: $O(V + E)$
Warning: Applicable only when the weight of the edges $\in \{0, x\}$

```
vector<pair<ll, int>> adj[maxn];
ll dists[maxn];
int s, n;
void bfs_01() {
    fill(dists, dists + n, oo);
    dist[s] = 0;
    deque<int> q;
    q.emplace_back(s);
    while (not q.empty()) {
        auto u = q.front();
        q.pop_front();
        for (auto [v, w] : adj[u]) {
            if (dist[v] <= dist[u] + w) continue;
            dist[v] = dist[u] + w;
            w ? q.emplace_back(v) : q.emplace_front(v);
        }
    }
}
```

8.4 Bellman ford

Description: Find shortest path from a single source to all other nodes. Can detect negative cycles.
Time: $O(V \cdot E)$

```
bool bellman_ford(const vector<vector<pair<int, ll>>> &g, int s,
                  vector<ll> &dist) {
    int n = (int)g.size();
    dist.assign(n, LLONG_MAX);
    vector<int> count(n);
    vector<char> in_queue(n);
    queue<int> q;
    dist[s] = 0;
    q.push(s);
    in_queue[s] = true;
    while (not q.empty()) {
        int cur = q.front();
        q.pop();
        in_queue[cur] = false;
        for (auto [to, w] : g[cur]) {
            if (dist[cur] + w < dist[to]) {
                dist[to] = dist[cur] + w;
                if (not in_queue[to]) {
                    q.push(to);
                    in_queue[to] = true;
                    count[to]++;
                    if (count[to] > n) return false;
                }
            }
        }
    }
    return true;
}
```

8.5 Bellman-Ford (find negative cycle)

Description: Given a directed graph find a negative cycle by running n iterations, and if the last one produces a relaxation than there is a cycle.
Time: $O(V \cdot E)$

```
const ll oo = 2500 * 1e9;
using graph = vector<vector<pair<int, ll>>>;
vi negative_cycle(graph &g, int n) {
    vll d(n, oo);
    vi p(n, -1);
    int x = -1;
    d[0] = 0;
    for (int i = 0; i < n; i++) {
        x = -1;
        for (int u = 0; u < n; u++) {
            for (auto &[v, l] : g[u]) {
                if (d[u] + l < d[v]) {
                    d[v] = d[u] + l;
                    p[v] = u;
                    x = v;
                }
            }
        }
    }
    if (x == -1)
        return {};
    else {
        for (int i = 0; i < n; i++) x = p[x];
        vi cycle;
        for (int v = x;; v = p[v]) {
            cycle.eb(v);
            if (v == x and len(cycle) > 1) break;
        }
        reverse(all(cycle));
        return cycle;
    }
}
```

8.6 Biconnected Components

Description: Build a vector of vectors, where the i-th vector correspond to the nodes of the i-th, biconnected component, a biconnected component is a subset of nodes and edges in which there is no cut point, also exist at least two distinct routes in vertex between any two vertex in the same biconnected component.
Time: $O(N + M)$

```
const int maxn(5 '00' 000);
int tin[maxn], stck[maxn], bcc_cnt, n, top = 0, timer = 1;
vector<int> g[maxn], nodes[maxn];
```

```cpp
int tarjan(int u, int p = -1) {
    int lowu = tin[u] = timer++;
    int son_cnt = 0;
    stck[++top] = u;
    for (auto v : g[u]) {
        if (!tin[v]) {
            son_cnt++;
            int lowx = tarjan(v, u);
            lowu = min(lowu, lowx);
            if (lowx >= tin[u]) {
                while (top != -1 && stck[top + 1] != v)
                    nodes[bcc_cnt].emplace_back(stck[top--]);
                nodes[bcc_cnt++].emplace_back(u);
            }
        } else {
            lowu = min(lowu, tin[v]);
        }
    }
    if (p == -1 && son_cnt == 0) {
        nodes[bcc_cnt++].emplace_back(u);
    }
    return lowu;
}
void build_bccs() {
    timer = 1;
    top = -1;
    memset(tin, 0, sizeof(int) * n);
    for (int i = 0; i < n; i++) nodes[i] = {};
    bcc_cnt = 0;
    for (int u = 0; u < n; u++)
        if (!tin[u]) tarjan(u);
}
```

8.7 Binary Lifting/Jumping

Description: Given a function/successor grpah answers queries of the form which is the node after k moves starting from u.
Time: Build $O(N \cdot MAXLOG2)$, Query $O(MAXLOG2)$.

```cpp
const int MAXN(2e5), MAXLOG2(30);
int bl[MAXN][MAXLOG2 + 1];
int N;
int jump(int u, ll k) {
    for (int i = 0; i <= MAXLOG2; i++) {
        if (k & (1ll << i)) u = bl[u][i];
    }
    return u;
}
void build() {
    for (int i = 1; i <= MAXLOG2; i++) {
        for (int j = 0; j < N; j++) {
            bl[j][i] = bl[bl[j][i - 1]][i - 1];
        }
    }
}
```

8.8 Bipartite Graph

Description: Given a graph, find the 'left' and 'right' side if is a bipartite graph, if is not then a empty vi2d is returned
Time: $O(N + M)$

```cpp
vi2d bipartite_graph(vi2d &adj) {
    int n = len(adj);
    vi side(n, -1);
    vi2d ret(2);
    rep(u, 0, n) {
        if (side[u] == -1) {
            queue<int> q;
            q.emp(u);
            side[u] = 0;
            ret[0].eb(u);
            while (len(q)) {
                int u = q.front();
                q.pop();
                for (auto v : adj[u]) {
                    if (side[v] == -1) {
                        side[v] = side[u] ^ 1;
                        ret[side[v]].eb(v);
                        q.push(v);
                    } else if (side[u] == side[v])
                        return {};
                }
            }
        }
    }
    return ret;
}
```

8.9 Block-Cut Tree * *

Description: Builds the Block-Cut of a undirected graph. * *
Usage: $isGraphCutpoint[u]$ answers how many connected components * are created when the node u is removed from the graph, if * $isGraphCutpoint[u]$ is greater than 1, it means that u is a * cutpoint. * *
Time: $O(N + M)$ * *
Memory: $O(N)$ * *
Warning: Always careful with disconnected graphs ! you may end up having * multiple trees. * *

```cpp
#pragma once
#include "../Contest/template.cpp"
struct BlockCutTree {
    int n;
    vi idOnTree, tin, low, stk, isGraphCutpoint, isTreeCutpoint;
    vi2d comps, treeAdj;
```

```cpp
    BlockCutTree(vi2d &g)
        : n(len(g)), idOnTree(n), tin(n), low(n), isGraphCutpoint(n) {
        rep(i, 0, n) {
            if (!tin[i]) {
                int timer = 0;
                dfs(i, -1, timer, g);
            }
        }
        buildTree();
    }
    void buildTree() {
        int node_id = 0;
        rep(u, 0, n) {
            if (isGraphCutpoint[u]) {
                idOnTree[u] = node_id++;
                isTreeCutpoint.eb(true);
                treeAdj.pb({});
            }
        }
        for (auto &comp : comps) {
            int node = node_id++;
            treeAdj.pb({});
            isTreeCutpoint.eb(false);
            for (int u : comp) {
                if (!isGraphCutpoint[u]) {
                    idOnTree[u] = node;
                } else {
                    treeAdj[node].eb(idOnTree[u]),
                        treeAdj[idOnTree[u]].eb(node);
                }
            }
        }
    }
    void dfs(int u, int p, int &timer, vi2d &g) {
        tin[u] = low[u] = ++timer;
        stk.eb(u);
        for (auto v : g[u]) {
            if (v == p) continue;
            if (!tin[v]) {
                dfs(v, u, timer, g);
                chmin(low[u], low[v]);
                if (low[v] >= tin[u]) {
                    isGraphCutpoint[u] += (tin[u] > 1 or tin[v] > 2);
                    comps.pb({u});
                    while (comps.back().back() != v) {
                        comps.back().eb(stk.back());
                        stk.ppb();
                    }
                }
            } else
                low[u] = min(low[u], tin[v]);
        }
    }
    int countMandatoryNodesOnPath(int startNode, int endNode);
};
```

8.10 Centroid Decomposition

Description: Builds a vector fat where fat_i is who is the father of the node i in the centroid decomposed tree.

```cpp
#pragma once
#include "../Contest/template.cpp"
vi centroidDecomposition(const vi2d &g) {
    int n = len(g);
    vi fat(n, -1), szt(n), tk(n);
    function<int(int, int)> calcsz = [&](int x, int f) {
        szt[x] = 1;
        for (auto y : g[x])
            if (y != f && !tk[y]) szt[x] += calcsz(y, x);
        return szt[x];
    };
    function<void(int, int, int)> cdfs = [&](int x, int f, int sz) {
        if (sz < 0) sz = calcsz(x, -1);
        for (auto y : g[x])
            if (!tk[y] && szt[y] * 2 >= sz) {
                szt[x] = 0;
                cdfs(y, f, sz);
                return;
            }
        tk[x] = true;
        fat[x] = f;
        for (auto y : g[x])
            if (!tk[y]) cdfs(y, x, -1);
    };
    cdfs(0, -1, -1);
    return fat;
}
```

8.11 Count mandatory nodes on a single path * *

Description: Given a $startNode$ and an $endNode$, count the mandatory nodes * in the path from $startNode$ to $endNode$, that is the number of nodes such * that are present in every possible such path. * *
Time: $O(N + M)$ * *
Memory: $O(N)$ * *
Warning: The $startNode$ and $endNode$ is always included in the counting, * ajust your final answer depending on the problem. Be careful with a * **disconnected graph** where the path may not exist, treat it appart !. * *

```cpp
#pragma once
#include "../Contest/template.cpp"
#include "./Block-Cut tree.cpp"
int BlockCutTree::countMandatoryNodesOnPath(int startNode, int endNode) {
    startNode = idOnTree[startNode], endNode = idOnTree[endNode];
    int ans = !isTreeCutpoint[startNode] + !isTreeCutpoint[endNode];
    int artPoints = 0;
```

```
    function<void(int, int)> dfsCount = [&](int u, int p) {
        artPoints += isTreeCutpoint[u];
        if (u == endNode) ans += artPoints;
        for (auto v : treeAdj[u]) {
            if (v != p) {
                dfsCount(v, u);
            }
        }
        artPoints -= isTreeCutpoint[u];
    };
    dfsCount(startNode, -1);
    return ans;
}
```

8.12 DSU query

```
struct DSU {
    V<ii> p;
    V<int> s;
    int sum = 0;
    DSU(int n) : p(n, {-1, -1}), s(n, 1) {}
    int find(int x) {
        if (p[x].ff < 0) return x;
        return find(p[x].ff);
    }
    void join(int x, int y, int w) {
        x = find(x);
        y = find(y);
        if (x == y) return;
        sum += w;
        if (s[x] < s[y]) swap(x, y);
        s[x] += s[y];
        p[y] = mp(x, w);
    }
    int query(int x, int y) {
        int r = 0;
        while (x != y) {
            if (s[x] < s[y])
                r = max(r, p[x].ss), x = p[x].ff;
            else
                r = max(r, p[y].ss), y = p[y].ff;
        }
        return r;
    }
};
```

8.13 D‘Escopo-Pape

Description: Is a single source shortest path that works faster than Dijkstra's algorithm and the Bellman-Ford algorithm in most cases, and will also work for negative edges. However not for negative cycles. There exists cases where it runs in exponential time.

Usage: Returns a pair containing two vectors, the first one with the distance from s to every other node, and another one with the ancestor of each node, note that the ancestor of s is -1

```
using Edge = pair<ll, int>;
using Adj = vector<vector<Edge>>;
pair<vll, vi> desopo_pape(int s, int n, const Adj &adj) {
    vll ds(n, LLONG_MAX), ps(n, -1);
    ds[s] = 0;
    vi ms(n, 2);
    deque<int> q;
    q.eb(s);
    while (len(q)) {
        int u = q.front();
        q.pop_front();
        ms[u] = 0;
        for (auto [w, v] : adj[u]) {
            if (chmin(ds[v], w + ds[u])) {
                ps[v] = u;
                if (ms[v] == 2)
                    ms[v] = 1, q.pb(v);
                else if (ms[v] == 0)
                    ms[v] = 1, q.pf(v);
            }
        }
    }
    return {ds, ps};
}
```

8.14 Dijkstra

```
const int MAXN = 1'00'000;
const ll MAXW = 1'000'000ll;
constexpr ll OO = MAXW * MAXN + 1;
using Edge = pair<ll, int>;  // { weigth, node}
using Adj = vector<vector<Edge>>;
template <typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;
pair<vll, vi> dijkstra(const Adj &g, int s) {
    int n = len(g);
    min_heap<Edge> pq;
    vll ds(n, OO);
    vi ps(n, -1);
    pq.emp(0, s);
    ds[s] = 0;
    while (len(pq)) {
        auto [du, u] = pq.top();
        pq.pop();
        if (ds[u] < du) continue;
        for (auto [w, v] : g[u]) {
            ll ndv = du + w;
            if (chmin(ds[v], ndv)) {
                ps[v] = u;
                pq.emp(ndv, v);
```

```
            }
        }
        return {ds, ps};
    }
    // optional !
    vi recover_path(int source, int ending, const vi &ps) {
        if (ps[ending] == -1) return {};
        int cur = ending;
        vi ans;
        while (cur != -1) {
            ans.eb(cur);
            cur = ps[cur];
        }
        reverse(all(ans));
        return ans;
    }
```

8.15 Dijkstra (K-shortest pahts)

```
const ll oo = 1e9 * 1e5 + 1;
using adj = vector<vector<pll>>;
vector<priority_queue<ll>> dijkstra(const vector<vector<pll>> &g, int n,
    int s,
                                    int k) {
    priority_queue<pll, vector<pll>, greater<pll>> pq;
    vector<priority_queue<ll>> dist(n);
    dist[0].emplace(0);
    pq.emplace(0, s);
    while (!pq.empty()) {
        auto [d1, v] = pq.top();
        pq.pop();
        if (not dist[v].empty() and dist[v].top() < d1) continue;
        for (auto [d2, u] : g[v]) {
            if (len(dist[u]) < k) {
                pq.emplace(d2 + d1, u);
                dist[u].emplace(d2 + d1);
            } else {
                if (dist[u].top() > d1 + d2) {
                    dist[u].pop();
                    dist[u].emplace(d1 + d2);
                    pq.emplace(d2 + d1, u);
                }
            }
        }
    }
    return dist;
}
```

8.16 Extra Edges to Make Digraph Fully Strongly Connected

Description: Given a directed graph G find the necessary edges to add to make the graph a single strongly connected component.

Time: $O(N + M)$

Memory: $O(N)$

```
struct SCC {
    int n, num_sccs;
    vi2d adj;
    vi scc_id;
    SCC(int _n) : n(_n), num_sccs(0), adj(n), scc_id(n, -1) {}
    SCC(const vi2d &_adj) : SCC(len(_adj)) {
        adj = _adj;
        find_sccs();
    }
    void add_edge(int u, int v) { adj[u].eb(v); }
    void find_sccs() {
        int timer = 1;
        vi tin(n), st;
        st.reserve(n);
        function<int(int)> dfs = [&](int u) -> int {
            int low = tin[u] = timer++, siz = len(st);
            st.eb(u);
            for (int v : adj[u])
                if (scc_id[v] < 0) low = min(low, tin[v] ? tin[v] : dfs(v)
    );
            if (tin[u] == low) {
                rep(i, siz, len(st)) scc_id[st[i]] = num_sccs;
                st.resize(siz);
                num_sccs++;
            }
            return low;
        };
        for (int i = 0; i < n; i++)
            if (!tin[i]) dfs(i);
    }
};
vector<array<int, 2>> extra_edges(const vi2d &adj) {
    SCC scc(adj);
    auto scc_id = scc.scc_id;
    auto num_sccs = scc.num_sccs;
    if (num_sccs == 1) return {};
    int n = len(adj);
    vi2d scc_adj(num_sccs);
    vi zero_in(num_sccs, 1);
    rep(u, 0, n) {
        for (int v : adj[u]) {
            if (scc_id[u] == scc_id[v]) continue;
            scc_adj[scc_id[u]].eb(scc_id[v]);
            zero_in[scc_id[v]] = 0;
        }
    }
    int random_source = max_element(all(zero_in)) - zero_in.begin();
    vi vis(num_sccs);
    function<int(int)> dfs = [&](int u) {
```

```
        if (empty(scc_adj[u])) return u;
        for (int v : scc_adj[u])
            if (!vis[v]) {
                vis[v] = 1;
                int zero_out = dfs(v);
                if (zero_out != -1) return zero_out;
            }
        return (int)-1;
    };
    vector<array<int, 2>> edges;
    vi in_unused;
    rep(i, 0, num_sccs) {
        if (zero_in[i]) {
            vis[i] = 1;
            int zero_out = dfs(i);
            if (zero_out != -1)
                edges.push_back({zero_out, i});
            else
                in_unused.push_back(i);
        }
    }
    rep(i, 1, len(edges)) { swap(edges[i][0], edges[i - 1][0]); }
    rep(i, 0, num_sccs) {
        if (scc_adj[i].empty() && !vis[i]) {
            if (!in_unused.empty()) {
                edges.push_back({i, in_unused.back()});
                in_unused.pop_back();
            } else {
                edges.push_back({i, random_source});
            }
        }
    }
    for (int u : in_unused) edges.push_back({0, u});
    vi to_node(num_sccs);
    rep(i, 0, n) to_node[scc_id[i]] = i;
    for (auto &[u, v] : edges) u = to_node[u], v = to_node[v];
    return edges;
}
```

8.17 Find Articulation/Cut Points

Description: Given an **undirected** graph find it's articulation points.
Time: $O(N + M)$
Warning: A vertex u can be an articulation point if and only if has at least 2 adjascent vertex

```
const int MAXN(100);
int N;
vi2d G;
int timer;
int tin[MAXN], low[MAXN];
set<int> cpoints;
int dfs(int u, int p = -1) {
    int cnt = 0;
    low[u] = tin[u] = timer++;
    for (auto v : G[u]) {
        if (not tin[v]) {
            cnt++;
            dfs(v, u);
            if (low[v] >= tin[u]) cpoints.insert(u);
            low[u] = min(low[u], low[v]);
        } else if (v != p)
            low[u] = min(low[u], tin[v]);
    }
    return cnt;
}
void getCutPoints() {
    memset(low, 0, sizeof(low));
    memset(tin, 0, sizeof(tin));
    cpoints.clear();
    timer = 1;
    for (int i = 0; i < N; i++) {
        if (tin[i]) continue;
        int cnt = dfs(i);
        if (cnt == 1) cpoints.erase(i);
    }
}
```

8.18 Find Bridge-Tree components

Usage: $label2CC(u, p)$ finds the 2-edge connected component of every node.
Time: $O(n + m)$

```
const int maxn(3 '00' 000);
int tin[maxn], compId[maxn], qtdComps;
vi g[maxn], stck;
int n;
int dfs(int u, int p = -1) {
    int low = tin[u] = len(stck);
    stck.emplace_back(u);
    bool multEdge = false;
    for (auto v : g[u]) {
        if (v == p and !multEdge) {
            multEdge = 1;
            continue;
        }
        low = min(low, tin[v] == -1 ? dfs(v, u) : tin[v]);
    }
    if (low == tin[u]) {
        for (int i = tin[u]; i < len(stck); i++) compId[stck[i]] =
    qtdComps;
        stck.resize(tin[u]);
        qtdComps++;
    }
    return low;
}
```

```cpp
void label2CC() {
    memset(compId, -1, sizeof(int) * n);
    memset(tin, -1, sizeof(int) * n);
    stck.reserve(n);
    for (int i = 0; i < n; i++) {
        if (tin[i] == -1) dfs(i);
    }
}
```

8.19 Find Bridges

Description: Find every bridge in a **undirected** connected graph.
Warning: Remember to read the graph as pair where the second is the id of the edge !
@Time : $O(N + M)$ const int MAXN(10000), MAXM(100000);

```cpp
int N, M, clk, tin[MAXN], low[MAXN], isBridge[MAXM];
vector<pii> G[MAXN];
void dfs(int u, int p = -1) {
    tin[u] = low[u] = clk++;
    for (auto [v, i] : G[u]) {
        if (v == p) continue;
        if (tin[v]) {
            low[u] = min(low[u], tin[v]);
        } else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > tin[u]) {
                isBridge[i] = 1;
            }
        }
    }
}
void findBridges() {
    fill(tin, tin + N, 0);
    fill(low, low + N, 0);
    fill(isBridge, isBridge + M, 0);
    clk = 1;
    for (int i = 0; i < N; i++) {
        if (!tin[i]) dfs(i);
    }
}
```

8.20 Find Centroid

Description: Given a tree (don't forget to make it 'undirected'), find it's centroids.
@Time : $O(V)$

```cpp
#pragma once
#include "../Contest/template.cpp"
void dfs(int u, int p, int n, vi2d &g, vi &sz, vi &centroid) {
    sz[u] = 1;
    bool iscentroid = true;
    for (auto v : g[u])
        if (v != p) {
            dfs(v, u, n, g, sz, centroid);
            if (sz[v] > n / 2) iscentroid = false;
            sz[u] += sz[v];
        }
    if (n - sz[u] > n / 2) iscentroid = false;
    if (iscentroid) centroid.eb(u);
}
vi getCentroid(vi2d &g, int n) {
    vi centroid;
    vi sz(n);
    dfs(0, -1, n, g, sz, centroid);
    return centroid;
}
```

8.21 Find bridges (online)

```cpp
// O((n+m)*log(n))
struct BridgeFinder {
    // 2ecc = 2 edge conected component
    // cc = conected component
    vector<int> parent, dsu_2ecc, dsu_cc, dsu_cc_size;
    int bridges, lca_iteration;
    vector<int> last_visit;
    BridgeFinder(int n)
        : parent(n, -1),
          dsu_2ecc(n),
          dsu_cc(n),
          dsu_cc_size(n, 1),
          bridges(0),
          lca_iteration(0),
          last_visit(n) {
        for (int i = 0; i < n; i++) {
            dsu_2ecc[i] = i;
            dsu_cc[i] = i;
        }
    }
    int find_2ecc(int v) {
        if (v == -1) return -1;
        return dsu_2ecc[v] == v ? v : dsu_2ecc[v] = find_2ecc(dsu_2ecc[v])
    ;
    }
    int find_cc(int v) {
        v = find_2ecc(v);
        return dsu_cc[v] == v ? v : dsu_cc[v] = find_cc(dsu_cc[v]);
    }
    void make_root(int v) {
        v = find_2ecc(v);
        int root = v;
        int child = -1;
        while (v != -1) {
            int p = find_2ecc(parent[v]);
```

```
        parent[v] = child;
        dsu_cc[v] = root;
        child = v;
        v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}
void merge_path(int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b;
    int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a);
            path_a.push_back(a);
            if (last_visit[a] == lca_iteration) {
                lca = a;
                break;
            }
            last_visit[a] = lca_iteration;
            a = parent[a];
        }
        if (b != -1) {
            b = find_2ecc(b);
            path_b.push_back(b);
            if (last_visit[b] == lca_iteration) {
                lca = b;
                break;
            }
            last_visit[b] = lca_iteration;
            b = parent[b];
        }
    }
    for (auto v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca) break;
        --bridges;
    }
    for (auto v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca) break;
        --bridges;
    }
}
void add_edge(int a, int b) {
    a = find_2ecc(a);
    b = find_2ecc(b);
    if (a == b) return;
    int ca = find_cc(a);
    int cb = find_cc(b);
    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b);
            swap(ca, cb);
        }
        make_root(a);
        parent[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else {
        merge_path(a, b);
    }
  }
};
```

8.22 Floyd Warshall

Description: Simply finds the minimal distance for each node to every other node. $O(V^3)$

```
vector<vll> floyd_warshall(const vector<vll> &adj, ll n) {
    auto dist = adj;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                dist[j][k] = min(dist[j][k], dist[j][i] + dist[i][k]);
            }
        }
    }
    return dist;
}
```

8.23 Functional/Successor Graph

Description: Given a functional graph find the vertice after k moves starting at u and also the distance between u and v, if it's impossible to reach v starting at u returns -1.
Time: build $O(N \cdot MAXLOG2)$, kth $O(MAXLOG2)$, dist $O(MAXLOG2)$

```
const int MAXN(2 '000' 000), MAXLOG2(24);
int N;
vi2d succ(MAXN, vi(MAXLOG2 + 1));
vi dst(MAXN, 0);
int vis[MAXN];
void dfsbuild(int u) {
    if (vis[u]) return;
    vis[u] = 1;
    int v = succ[u][0];
    dfsbuild(v);
    dst[u] = dst[v] + 1;
}
void build() {
    for (int i = 0; i < N; i++) {
        if (not vis[i]) dfsbuild(i);
    }
    for (int k = 1; k <= MAXLOG2; k++) {
        for (int i = 0; i < N; i++) {
            succ[i][k] = succ[succ[i][k - 1]][k - 1];
        }
    }
```

```
}
int kth(int u, ll k) {
    if (k <= 0) return u;
    for (int i = 0; i <= MAXLOG2; i++)
        if ((1ll << i) & k) u = succ[u][i];
    return u;
}
int dist(int u, int v) {
    int cu = kth(u, dst[u]);
    if (kth(u, dst[u] - dst[v]) == v)
        return dst[u] - dst[v];
    else if (kth(cu, dst[cu] - dst[v]) == v)
        return dst[u] + (dst[cu] - dst[v]);
    else
        return -1;
}
```

8.24 Heavy light decomposition (supreme)

```
struct HLD {
    int V;
    int id;
    int nb_heavy_path;
    std::vector<std::vector<int>> g;
    std::vector<pair<int, int>> edges;  // edges of the tree
    std::vector<int> par;               // par[i] = parent of
                                        // vertex i (Default: -1)
    std::vector<int> depth;             // depth[i] = distance between
    root
                                        // and vertex i
    std::vector<int> subtree_sz;        // subtree_sz[i] = size of
                                        // subtree whose root is i
    std::vector<int> heavy_child;       // heavy_child[i] = child of
                                        // vertex i on heavy path
                                        // (Default: -1)
    std::vector<int> tree_id;           // tree_id[i] = id of tree vertex
                                        // i belongs to
    std::vector<int> aligned_id,
        aligned_id_inv;             // aligned_id[i] =  aligned
                                    // id for vertex i
                                    // (consecutive on heavy
                                    // edges)
    std::vector<int> head;          // head[i] = id of vertex on heavy
                                    // path of vertex i, nearest to root
    std::vector<int> head_ids;      // consist of head vertex id's
    std::vector<int> heavy_path_id; // heavy_path_id[i] =
                                    // heavy_path_id for vertex
                                    // [i]

    HLD(const std::vector<std::vector<int>> &e, vector<int> roots = {0})
        : HLD((int)e.size()) {
        g = e;
        build(roots);
    }
    HLD(int sz = 0)
        : V(sz),
          id(0),
          nb_heavy_path(0),
          g(sz),
          par(sz),
          depth(sz),
          subtree_sz(sz),
          heavy_child(sz),
          tree_id(sz, -1),
          aligned_id(sz),
          aligned_id_inv(sz),
          head(sz),
          heavy_path_id(sz, -1) {}
    void add_edge(int u, int v) {
        edges.emplace_back(u, v);
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    void _build_dfs(int root) {
        std::stack<std::pair<int, int>> st;
        par[root] = -1;
        depth[root] = 0;
        st.emplace(root, 0);
        while (!st.empty()) {
            int now = st.top().first;
            int &i = st.top().second;
            if (i < (int)g[now].size()) {
                int nxt = g[now][i++];
                if (nxt == par[now]) continue;
                par[nxt] = now;
                depth[nxt] = depth[now] + 1;
                st.emplace(nxt, 0);
            } else {
                st.pop();
                int max_sub_sz = 0;
                subtree_sz[now] = 1;
                heavy_child[now] = -1;
                for (auto nxt : g[now]) {
                    if (nxt == par[now]) continue;
                    subtree_sz[now] += subtree_sz[nxt];
                    if (max_sub_sz < subtree_sz[nxt])
                        max_sub_sz = subtree_sz[nxt], heavy_child[now] =
nxt;
                }
            }
        }
    }
    void _build_bfs(int root, int tree_id_now) {
        std::queue<int> q({root});
        while (!q.empty()) {
            int h = q.front();
            q.pop();
            head_ids.emplace_back(h);
            for (int now = h; now != -1; now = heavy_child[now]) {
                tree_id[now] = tree_id_now;
```

```cpp
                aligned_id[now] = id++;
                aligned_id_inv[aligned_id[now]] = now;
                heavy_path_id[now] = nb_heavy_path;
                head[now] = h;
                for (int nxt : g[now])
                    if (nxt != par[now] and nxt != heavy_child[now])
                        q.push(nxt);
            }
            nb_heavy_path++;
        }
    }
    void build(std::vector<int> roots = {0}) {
        int tree_id_now = 0;
        for (auto r : roots) _build_dfs(r), _build_bfs(r, tree_id_now++);
    }
    // data[i] = value of vertex i
    template <class T>
    std::vector<T> segtree_rearrange(const std::vector<T> &data) const {
        assert(int(data.size()) == V);
        std::vector<T> ret;
        ret.reserve(V);
        for (int i = 0; i < V; i++) ret.emplace_back(data[aligned_id_inv[i
]]);
        return ret;
    }
    // data[i] = weight of edge[i]
    template <class T>
    std::vector<T> segtree_rearrange_weighted(
        const std::vector<T> &data) const {
        assert(data.size() == edges.size());
        vector<T> ret(V);
        for (int i = 0; i < (int)edges.size(); i++) {
            auto [u, v] = edges[i];
            if (depth[u] > depth[v]) swap(u, v);
            ret[aligned_id[v]] = data[i];
        }
        return ret;
    }
    int segtree_edge_index(int i) const {
        auto [u, v] = edges[i];
        if (depth[u] > depth[v]) swap(u, v);
        return aligned_id[v];
    }
    // query for vertices on path [u, v] (INCLUSIVE)
    void for_each_vertex(int u, int v, const auto &f) const {
        static_assert(std::is_invocable_r_v<void, decltype(f), int, int>);
        assert(tree_id[u] == tree_id[v] and tree_id[u] >= 0);
        while (true) {
            if (aligned_id[u] > aligned_id[v]) std::swap(u, v);
            f(std::max(aligned_id[head[v]], aligned_id[u]), aligned_id[v])
;
            if (head[u] == head[v]) break;
            v = par[head[v]];
        }
    }
    void for_each_vertex_noncommutative(int from, int to, const auto &fup,
                                        const auto &fdown) const {
        static_assert(std::is_invocable_r_v<void, decltype(fup), int, int
>);
        static_assert(std::is_invocable_r_v<void, decltype(fdown), int,
int>);
        assert(tree_id[from] == tree_id[to] and tree_id[from] >= 0);
        int u = from, v = to;
        const int lca = lowest_common_ancestor(u, v), dlca = depth[lca];
        while (u >= 0 and depth[u] > dlca) {
            const int p = (depth[head[u]] > dlca ? head[u] : lca);
            fup(aligned_id[p] + (p == lca), aligned_id[u]), u = par[p];
        }
        static std::vector<std::pair<int, int>> lrs;
        int sz = 0;
        while (v >= 0 and depth[v] >= dlca) {
            const int p = (depth[head[v]] >= dlca ? head[v] : lca);
            if (int(lrs.size()) == sz) lrs.emplace_back(0, 0);
            lrs.at(sz++) = {p, v}, v = par.at(p);
        }
        while (sz--)
            fdown(aligned_id[lrs.at(sz).first], aligned_id[lrs.at(sz).
second]);
    }
    // query for edges on path [u, v]
    void for_each_edge(int u, int v, const auto &f) const {
        static_assert(std::is_invocable_r_v<void, decltype(f), int, int>);
        assert(tree_id[u] == tree_id[v] and tree_id[u] >= 0);
        while (true) {
            if (aligned_id[u] > aligned_id[v]) std::swap(u, v);
            if (head[u] != head[v]) {
                f(aligned_id[head[v]], aligned_id[v]);
                v = par[head[v]];
            } else {
                if (u != v) f(aligned_id[u] + 1, aligned_id[v]);
                break;
            }
        }
    }
    // lowest_common_ancestor: O(log V)
    int lowest_common_ancestor(int u, int v) const {
        assert(tree_id[u] == tree_id[v] and tree_id[u] >= 0);
        while (true) {
            if (aligned_id[u] > aligned_id[v]) std::swap(u, v);
            if (head[u] == head[v]) return u;
            v = par[head[v]];
        }
    }
    int distance(int u, int v) const {
        assert(tree_id[u] == tree_id[v] and tree_id[u] >= 0);
        return depth[u] + depth[v] - 2 * depth[lowest_common_ancestor(u, v
)];
    }
    // Level ancestor, O(log V)
```

```
    // if k-th parent is out of range, return -1
    int kth_parent(int v, int k) const {
        if (k < 0) return -1;
        while (v >= 0) {
            int h = head.at(v), len = depth.at(v) - depth.at(h);
            if (k <= len) return aligned_id_inv.at(aligned_id.at(v) - k);
            k -= len + 1, v = par.at(h);
        }
        return -1;
    }
    // Jump on tree, O(log V)
    int s_to_t_by_k_steps(int s, int t, int k) const {
        if (k < 0) return -1;
        if (k == 0) return s;
        int lca = lowest_common_ancestor(s, t);
        if (k <= depth.at(s) - depth.at(lca)) return kth_parent(s, k);
        return kth_parent(t, depth.at(s) + depth.at(t) - depth.at(lca) * 2
     - k);
    }
};
```

8.25 Kruskal

Description: Find the minimum spanning tree of a graph.
Time: $O(E \log E)$

```
#include "./../Data Structures/DSU.cpp"
vector<tuple<ll, int, int>> kruskal(int n, vector<tuple<ll, int, int>> &
    edges) {
    DSU dsu(n);
    vector<tuple<ll, int, int>> ans;
    sort(all(edges));
    for (auto [a, b, c] : edges) {
        if (dsu.same_set(b, c)) continue;
        ans.emplace_back(a, b, c);
        dsu.union_set(b, c);
    }
    return ans;
}
```

8.26 Lowest Common Ancestor

Description: Given two nodes of a tree find their lowest common ancestor, or their distance

```
#pragma once
#include "../Contest/template.cpp"
template <typename T>
struct SparseTable {
    vector<T> v;
    int n;
    static const int b = 30;
    vi mask, t;
    int op(int x, int y) { return v[x] < v[y] ? x : y; }
    int msb(int x) { return __builtin_clz(1) - __builtin_clz(x); }
    SparseTable() {}
    SparseTable(const vector<T> &v_) : v(v_), n(v.size()), mask(n), t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
            at = (at << 1) & ((1 << b) - 1);
            while (at and op(i, i - msb(at & -at)) == i) at ^= at & -at;
        }
        for (int i = 0; i < n / b; i++)
            t[i] = b * i + b - 1 - msb(mask[b * i + b - 1]);
        for (int j = 1; (1 << j) <= n / b; j++)
            for (int i = 0; i + (1 << j) <= n / b; i++)
                t[n / b * j + i] = op(t[n / b * (j - 1) + i],
                                      t[n / b * (j - 1) + i + (1 << (j -
1))]);
    }
    int small(int r, int sz = b) { return r - msb(mask[r] & ((1 << sz) -
    1)); }
    T query(int l, int r) {
        if (r - l + 1 <= b) return small(r, r - l + 1);
        int ans = op(small(l + b - 1), small(r));
        int x = l / b + 1, y = r / b - 1;
        if (x <= y) {
            int j = msb(y - x + 1);
            ans =
                op(ans, op(t[n / b * j + x], t[n / b * j + y - (1 << j) +
1]));
        }
        return ans;
    }
};
struct LCA {
    SparseTable<int> st;
    int n;
    vi v, pos, dep;
    vll wdep;
    LCA(const Graph &g, int root) : n(len(g)), pos(n), wdep(n) {
        dfs(root, 0, -1, g);
        st = SparseTable<int>(vector<int>(all(dep)));
    }
    void dfs(int i, int d, int p, const Graph &g) {
        v.eb(len(dep)) = i, pos[i] = len(dep), dep.eb(d);
        for (auto [w, j] : g[i])
            if (j != p) {
                wdep[j] = wdep[i] + w;
                dfs(j, d + 1, i, g);
                v.eb(len(dep)) = i, dep.eb(d);
            }
    }
    int lca(int a, int b) {
        int l = min(pos[a], pos[b]);
        int r = max(pos[a], pos[b]);
        return v[st.query(l, r)];
    }
    ll dist(int a, int b) { return wdep[a] + wdep[b] - 2ll * wdep[lca(a, b
```

```
    )]; }
};
```

8.27 Lowest Common Ancestor (Binary Lifting)

Description: Given a directed tree, finds the LCA between two nodes using binary lifting, and answer a few queries with it.
Usage:

- lca: returns the LCA between the two given nodes
- on_path: fids if c is in the path from a to b

Time: build $O(N \cdot MAXLOG2)$, all queries $O(MAXLOG2)$

```
struct LCA {
    int n;
    const int maxlog;
    vector<vector<int>> up;
    vector<int> depth;
    LCA(const vector<vector<int>> &tree)
        : n(tree.size()),
          maxlog(ceil(log2(n))),
          up(n, vector<int>(maxlog + 1)),
          depth(n, -1) {
        for (int i = 0; i < n; i++) {
            if (depth[i] == -1) {
                depth[i] = 0;
                dfs(i, -1, tree);
            }
        }
    }
    void dfs(int u, int p, const vector<vector<int>> &tree) {
        if (p != -1) {
            depth[u] = depth[p] + 1;
            up[u][0] = p;
            for (int i = 1; i <= maxlog; i++) {
                up[u][i] = up[up[u][i - 1]][i - 1];
            }
        }
        for (int v : tree[u]) {
            if (v == p) continue;
            dfs(v, u, tree);
        }
    }
    int kth_jump(int u, int k) {
        for (int i = maxlog; i >= 0; i--) {
            if ((1 << i) & k) {
                u = up[u][i];
            }
        }
        return u;
    }
    int lca(int u, int v) {
        if (depth[u] < depth[v]) swap(u, v);
        int diff = depth[u] - depth[v];
        u = kth_jump(u, diff);
        if (u == v) return u;
        for (int i = maxlog; i >= 0; i--) {
            if (up[u][i] != up[v][i]) {
                u = up[u][i];
                v = up[v][i];
            }
        }
        return up[u][0];
    }
    bool on_path(int u, int v, int s) {
        int uv = lca(u, v), us = lca(u, s), vs = lca(v, s);
        return (uv == s or (us == uv and vs == s) or (vs == uv and us == s
));
    }
    int dist(int u, int v) {
        return depth[u] + depth[v] - 2 * depth[lca(u, v)];
    }
};
```

8.28 Maximum flow (Dinic)

Description: Finds the **maximum flow** in a graph network, given the **source** s and the **sink** t. Add edge from a to b with capcity c.
Time: In general $O(E \cdot V^2)$, if every capacity is 1, and every vertice has in degree equal 1 or out degree equal 1 then $O(E \cdot \sqrt{V})$,
Warning: Suffle the edges list for every vertice may take you out of the worst case

```
struct Dinic {
    struct Edge {
        int to, rev;
        ll c, oc;
        ll flow() { return max(oc - c, 0LL); }  // if you need flows
    };
    vi lvl, ptr, q;
    vector<vector<Edge>> adj;
    Dinic(int n) : lvl(n), ptr(n), q(n), adj(n) {}
    void addEdge(int a, int b, ll c, ll rcap = 0) {
        adj[a].pb({b, len(adj[b]), c, c});
        adj[b].pb({a, len(adj[a]) - 1, rcap, rcap});
    }
    ll dfs(int v, int t, ll f) {
        if (v == t || !f) return f;
        for (int &i = ptr[v]; i < len(adj[v]); i++) {
            Edge &e = adj[v][i];
            if (lvl[e.to] == lvl[v] + 1)
                if (ll p = dfs(e.to, t, min(f, e.c))) {
                    e.c -= p, adj[e.to][e.rev].c += p;
                    return p;
                }
        }
        return 0;
    }
```

```
    ll maxFlow(int s, int t) {
        ll flow = 0;
        q[0] = s;
        rep(L, 0, 31) {
            do {  // 'int L=30' maybe faster for random
                  // data
                lvl = ptr = vi(len(q));
                int qi = 0, qe = lvl[s] = 1;
                while (qi < qe && !lvl[t]) {
                    int v = q[qi++];
                    for (Edge e : adj[v])
                        if (!lvl[e.to] && e.c >> (30 - L))
                            q[qe++] = e.to, lvl[e.to] = lvl[v] + 1;
                }
                while (ll p = dfs(s, t, LLONG_MAX)) flow += p;
            } while (lvl[t]);
        }
        return flow;
    }
    bool leftOfMinCut(int a) { return lvl[a] != 0; }
};
```

8.29 Minimum Cost Flow

Description: Given a network find the minimum cost to achieve a flow of at most f. Works with **directed** and **undirected** graphs

Usage:

- **add(u, v, c, w):** adds an edge from u to v with capacity c and cost w.
- **flow(f):** return a pair $(flow, cost)$ with the maximum flow until f with source at s and sink at t, with the minimum cost possible.

Time: $O(N \cdot M + f \cdot m \log n)$

```
template <typename T>
struct MinCostFlow {
    struct Edge {
        int to;
        ll c, rc;  // capcity, residual capacity
        T w;       // cost
    };
    int n, s, t;
    vector<Edge> edges;
    vi2d g;
    vector<T> dist;
    vi pre;

    MinCostFlow() {}
    MinCostFlow(int n_, int _s, int _t) : n(n_), s(_s), t(_t), g(n) {}

    void addEdge(int u, int v, ll c, T w) {
        g[u].pb(len(edges));
        edges.eb(v, c, 0, w);
        g[v].pb(len(edges));
        edges.eb(u, 0, 0, -w);
    }

    // {flow, cost}
    pair<ll, T> flow(ll flow_limit = LLONG_MAX) {
        ll flow = 0;
        T cost = 0;
        while (flow < flow_limit and dijkstra(s, t)) {
            ll aug = LLONG_MAX;
            for (int i = t; i != s; i = edges[pre[i] ^ 1].to) {
                aug = min({flow_limit - flow, aug, edges[pre[i]].c});
            }
            for (int i = t; i != s; i = edges[pre[i] ^ 1].to) {
                edges[pre[i]].c -= aug;
                edges[pre[i] ^ 1].c += aug;

                edges[pre[i]].rc += aug;
                edges[pre[i] ^ 1].rc -= aug;
            }
            flow += aug;
            cost += (T)aug * dist[t];
        }
        return {flow, cost};
    }

    // Needs to be called after flow method
    vi2d paths() {
        vi2d p;
        for (;;) {
            int cur = s;
            auto &res = p.eb();
            res.pb(cur);
            while (cur != t) {
                bool found = false;
                for (auto i : g[cur]) {
                    auto &[v, _, c, cost] = edges[i];
                    if (c > 0) {
                        --c;
                        res.pb(cur = v);
                        found = true;
                        break;
                    }
                }
                if (!found) break;
            }
            if (cur != t) {
                p.ppb();
                break;
            }
        }
        return p;
    }

   private:
    bool bellman_ford(int s, int t) {
        dist.assign(n, numeric_limits<T>::max());
        pre.assign(n, -1);

        vc inq(n, false);
        queue<int> q;

        dist[s] = 0;
        q.push(s);
```

```
        while (len(q)) {
            int u = q.front();
            q.pop();
            inq[u] = false;
            for (int i : g[u]) {
                auto [v, c, w, _] = edges[i];
                auto new_dist = dist[u] + w;
                if (c > 0 and dist[v] > new_dist) {
                    dist[v] = new_dist;
                    pre[v] = i;
                    if (not inq[v]) {
                        inq[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        return dist[t] != numeric_limits<T>::max();
    }
    bool dijkstra(int s, int t) {
        dist.assign(n, numeric_limits<T>::max());
        pre.assign(n, -1);
        dist[s] = 0;
        using PQ = pair<T, int>;
        pqmn<PQ> pq;
        pq.emp(0, s);
        while (len(pq)) {
            auto [cost, u] = pq.top();
            pq.pop();
            if (cost != dist[u]) continue;
            for (int i : g[u]) {
                auto [v, c, _, w] = edges[i];
                auto new_dist = dist[u] + w;
                if (c > 0 and dist[v] > new_dist) {
                    dist[v] = new_dist;
                    pre[v] = i;
                    pq.emp(new_dist, v);
                }
            }
        }
        return dist[t] != numeric_limits<T>::max();
    }
};
```

8.30 Minimum Vertex Cover (already divided)

Description: Given a bipartite graph g with n vertices at left and m vertices at right, where $g[i]$ are the possible right side matches of vertex i from left side, find a minimum vertex cover. The size is the same as the size of the maximum matching, and the complement is a maximum independent set.

```
vector<int> min_vertex_cover(vector<vector<int>> &g, int n, int m) {
    vector<int> match(m, -1), vis;
    auto find = [&](auto &&self, int j) -> bool {
        if (match[j] == -1) return 1;
        vis[j] = 1;
        int di = match[j];
        for (int e : g[di])
            if (!vis[e] and self(self, e)) {
                match[e] = di;
                return 1;
            }
        return 0;
    };
    for (int i = 0; i < (int)g.size(); i++) {
        vis.assign(match.size(), 0);
        for (int j : g[i]) {
            if (find(find, j)) {
                match[j] = i;
                break;
            }
        }
    }
    int res = (int)match.size() - (int)count(match.begin(), match.end(),
-1);
    vector<char> lfound(n, true), seen(m);
    for (int it : match)
        if (it != -1) lfound[it] = false;
    vector<int> q, cover;
    for (int i = 0; i < n; i++)
        if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back();
        q.pop_back();
        lfound[i] = 1;
        for (int e : g[i])
            if (!seen[e] and match[e] != -1) {
                seen[e] = true;
                q.push_back(match[e]);
            }
    }
    for (int i = 0; i < n; i++)
        if (!lfound[i]) cover.push_back(i);
    for (int i = 0; i < m; i++)
        if (seen[i]) cover.push_back(n + i);
    assert((int)size(cover) == res);
    return cover;
}
```

8.31 Prim (MST)

Description: Given a graph with N vertex finds the minimum spanning tree, if there is no such three returns inf, it starts using the edges that connect with each $s_i \in s$, if none is provided than it starts with the edges of node 0.
Time: $O(V \log E)$

```cpp
#include "../Contest/template.cpp"
const int MAXN(1'00'000);
int N;
vector<pair<ll, int>> G[MAXN];
ll prim(vi s = vi(1, 0)) {
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll,
    int>>>
        pq;
    vector<char> ingraph(MAXN);
    int ingraphcnt(0);
    for (auto si : s) {
        ingraphcnt++;
        ingraph[si] = true;
        for (auto &[w, v] : G[si]) pq.emplace(w, v);
    }
    ll mstcost = 0;
    while (ingraphcnt < N and !pq.empty()) {
        ll w;
        int v;
        do {
            tie(w, v) = pq.top();
            pq.pop();
        } while (not pq.empty() and ingraph[v]);
        mstcost += w, ingraph[v] = true, ingraphcnt++;
        for (auto &[w2, v2] : G[v]) {
            pq.emplace(w2, v2);
        }
    }
    return ingraphcnt == N ? mstcost : oo;
}
```

8.32 Reachability Tree

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 20000 + 100;
int dsu[MAXN];
int n;
const int MAXM = 100000;
int m;
int U[MAXM], V[MAXM];
vector<int> adj[MAXN];
int getRoot(int u) {
    if (u == dsu[u]) return u;
    return dsu[u] = getRoot(dsu[u]);
}
void addEdge(int u, int v) {
    u = getRoot(u);
    v = getRoot(v);
    dsu[n] = n;
    dsu[u] = dsu[v] = n;
    adj[n].push_back(u);
    if (u != v) adj[n].push_back(v);
    ++n;
}
void build() {
    for (int i = 0; i < n; ++i) dsu[i] = i;
    for (int i = 0; i < m; ++i) addEdge(U[i], V[i]);
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
}
```

8.33 Shortest Path With K-edges

Description: Given an adjacency matrix of a graph, and a number K computes the shortest path between all nodes that uses exactly K edges, so for $0 \leq i, j \leq N-1$ ans[i][j] = "the shortest path between i and j that uses exactly K edges, remember to initialize the adjacency matrix with ∞.
Time: $O(N^3 \cdot \log K)$

```cpp
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a, vector<vector<T>> &b) {
    const T _oo = numeric_limits<T>::max();
    int n = a.size();
    vector<vector<T>> c(n, vector<T>(n, _oo));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                if (a[i][k] != _oo and b[k][j] != _oo)
                    c[i][j] = min(c[i][j], a[i][k] + b[k][j]);
    return c;
}
template <typename T>
vector<vector<T>> shortest_with_k_moves(vector<vector<T>> adj, long long k
    ) {
    if (k == 1) return adj;
    auto ans = adj;
    k--;
    while (k) {
        if (k & 1) ans = prod(ans, adj);
        k >>= 1;
        adj = prod(adj, adj);
    }
    return ans;
}
```

8.34 Strongly Connected Components (struct)

Description: Find the connected component for each edge (already in a topological order), some additional functions are also provided.

Time: Build: $O(V + E)$

```cpp
struct SCC {
    int n, num_sccs;
    vi2d adj;
    vi scc_id;
    SCC(int _n) : n(_n), num_sccs(0), adj(n), scc_id(n, -1) {}
    void add_edge(int u, int v) { adj[u].eb(v); }
    void find_sccs() {
        int timer = 1;
        vi tin(n), st;
        st.reserve(n);
        function<int(int)> dfs = [&](int u) -> int {
            int low = tin[u] = timer++, siz = len(st);
            st.eb(u);
            for (int v : adj[u])
                if (scc_id[v] < 0) low = min(low, tin[v] ? tin[v] : dfs(v)
    );
            if (tin[u] == low) {
                rep(i, siz, len(st)) scc_id[st[i]] = num_sccs;
                st.resize(siz);
                num_sccs++;
            }
            return low;
        };
        for (int i = 0; i < n; i++)
            if (!tin[i]) dfs(i);
    }
    vector<set<int>> build_gscc() {
        vector<set<int>> gscc;
        for (int i = 0; i < len(adj); ++i)
            for (auto j : adj[i])
                if (scc_id[i] != scc_id[j]) gscc[scc_id[i]].emplace(scc_id
    [j]);
        return gscc;
    }
    vi2d per_comp() {
        vi2d ret(num_sccs);
        rep(i, 0, n) ret[scc_id[i]].eb(i);
        reverse(all(ret));  // already in topological order ;)
        return ret;
    }
};
```

8.35 Topological Sorting (Kahn)

Description: Finds the topological sorting in a **DAG**, if the given graph is not a **DAG** than an empty vector is returned, need to 'initialize' the **INCNT** as you build the graph.
Time: $O(V + E)$

```cpp
const int MAXN(2 '00' 000);
int INCNT[MAXN];
vi2d GOUT(MAXN);
int N;
vi toposort() {
    vi order;
    queue<int> q;
    for (int i = 0; i < N; i++)
        if (!INCNT[i]) q.emplace(i);
    while (!q.empty()) {
        auto u = q.front();
        q.pop();
        order.emplace_back(u);
        for (auto v : GOUT[u]) {
            INCNT[v]--;
            if (INCNT[v] == 0) q.emplace(v);
        }
    }
    return len(order) == N ? order : vi();
}
```

8.36 Topological Sorting (Tarjan)

Description: Finds a the topological order for the graph, if there is no such order it means the graph is cyclic, then it returns an empty vector
Time: $O(V + E)$

```cpp
const int maxn(1 '00' 000);
int n, m;
vi g[maxn];
int not_found = 0, found = 1, processed = 2;
int state[maxn];
bool dfs(int u, vi &order) {
    if (state[u] == processed) return true;
    if (state[u] == found) return false;
    state[u] = found;
    for (auto v : g[u]) {
        if (not dfs(v, order)) return false;
    }
    state[u] = processed;
    order.emplace_back(u);
    return true;
}
vi topo_sort() {
    vi order;
    memset(state, 0, sizeof state);
    for (int u = 0; u < n; u++) {
        if (state[u] == not_found and not dfs(u, order)) return {};
    }
    reverse(all(order));
    return order;
}
```

8.37 Tree Isomorphism (not rooted)

Description: Two trees are considered **isomorphic** if the hash given by $thash()$ is the same.
Time: $O(V \cdot \log V)$

```
map<vi, int> mphash;
struct Tree {
    int n;
    vi2d g;
    vi sz, cs;
    Tree(int n_) : n(n_), g(n), sz(n) {}
    void add_edge(int u, int v) {
        g[u].emplace_back(v);
        g[v].emplace_back(u);
    }
    void dfs_centroid(int v, int p) {
        sz[v] = 1;
        bool cent = true;
        for (int u : g[v])
            if (u != p) {
                dfs_centroid(u, v);
                sz[v] += sz[u];
                cent &= not(sz[u] > n / 2);
            }
        if (cent and n - sz[v] <= n / 2) cs.push_back(v);
    }
    int fhash(int v, int p) {
        vi h;
        for (int u : g[v])
            if (u != p) h.push_back(fhash(u, v));
        sort(all(h));
        if (!mphash.count(h)) mphash[h] = mphash.size();
        return mphash[h];
    }
    ll thash() {
        cs.clear();
        dfs_centroid(0, -1);
        if (cs.size() == 1) return fhash(cs[0], -1);
        ll h1 = fhash(cs[0], cs[1]), h2 = fhash(cs[1], cs[0]);
        return (min(h1, h2) << 30ll) + max(h1, h2);
    }
};
```

8.38 Tree Isomorphism (rooted)

Description: Given a rooted tree find the hash of each subtree, if two roots of two distinct trees have the same hash they are considered isomorphic
Time: hash first time in $O(\log N_v \cdot N_v)$ where (N_v) is the of the subtree of v

```
map<vi, int> hasher;
int hs = 0;
struct RootedTreeIso {
    int n;
    vi2d adj;
    vi hashes;
    RootedTreeIso(int _n) : n(_n), adj(_n), hashes(_n, -1) {};
    void add_edge(int u, int v) {
        adj[u].emplace_back(v);
        adj[v].emplace_back(u);
    }
    int hash(int u, int p = -1) {
        if (hashes[u] != -1) return hashes[u];
        vi children;
        for (auto v : adj[u])
            if (v != p) children.emplace_back(hash(v, u));
        sort(all(children));
        if (!hasher.count(children)) hasher[children] = hs++;
        return hashes[u] = hasher[children];
    }
};
```

8.39 Tree diameter (DP)

```
const int MAXN(1 '000' 000);
int N;
vi G[MAXN];
int diameter, toLeaf[MAXN];
void calcDiameter(int u = 0, int p = -1) {
    int d1, d2;
    d1 = d2 = -1;
    for (auto v : G[u]) {
        if (v != p) {
            calcDiameter(v, u);
            d1 = max(d1, toLeaf[v]);
            tie(d1, d2) = minmax({d1, d2});
        }
    }
    toLeaf[u] = d2 + 1;
    diameter = max(diameter, d1 + d2 + 2);
}
```

8.40 Tree edge queries

```
template <typename T = ll, auto E = 0,
          auto F = [](ll a, ll b) { return max(a, b); }>
struct TEQ {
    const int LOG = 20;
    using Graph = vector<vector<pair<ll, int>>>;
    int n;
    vector<int> h;
    vector<vector<int>> par;
    vector<vector<T>> ed;
    TEQ(const Graph& g, int root = 0)
        : n(size(g)),
```

```cpp
        h(n, -1),
        par(n, vector<int>(LOG + 1, root)),
        ed(n, vector<T>(LOG + 1, E)) {
      h[root] = 0, dfs(root, g);
    }
    void dfs(int u, const Graph& g) {
        for (auto& [w, v] : g[u]) {
            if (h[v] == -1) {
                h[v] = h[u] + 1, par[v][0] = u, ed[v][0] = w;
                for (int k = 0, p; k < LOG; k++) {
                    p = par[v][k];
                    par[v][k + 1] = par[p][k];
                    ed[v][k + 1] = F(ed[v][k], ed[p][k]);
                }
                dfs(v, g);
            }
        }
    }
    pair<int, T> up(int u, int dis) {
        T res = E;
        for (int k = 0; k <= LOG; k++) {
            if (dis & (1 << k)) {
                res = F(res, ed[u][k]);
                u = par[u][k];
            }
        }
        return {u, res};
    }
    pair<int, T> lca(int u, int v) {
        if (h[u] > h[v]) swap(u, v);
        T res = E;
        tie(v, res) = up(v, h[v] - h[u]);
        if (v == u) return {v, res};
        for (int k = LOG; ~k; k--) {
            if (par[u][k] != par[v][k]) {
                res = F(res, ed[v][k]);
                res = F(res, ed[u][k]);
                u = par[u][k], v = par[v][k];
            }
        }
        res = F(res, ed[v][0]);
        res = F(res, ed[u][0]);
        return {par[v][0], res};
    }
};
```

8.41 Virtual Tree

```cpp
#pragma once
#include "../Contest/template.cpp"
#include "./Lowest common ancestor (sparse table).cpp"
struct VTree {
    int n;
    LCA lca;
    VTree(const Graph& g, int root = 0) : n(len(g)), lca(g, root) {}
    pair<vector<tuple<ll, int, int>>, int> vtree(vector<int> vs) {
        sort(vs.begin(), vs.end(),
             [&](int u, int v) { return lca.pos[u] < lca.pos[v]; });
        for (int i = 0, n = size(vs); i + 1 < n; i++) {
            vs.eb(lca.lca(vs[i], vs[i + 1]));
        }
        sort(vs.begin(), vs.end(),
             [&](int u, int v) { return lca.pos[u] < lca.pos[v]; });
        vs.erase(unique(all(vs)), vs.end());
        vi st{vs.front()};
        vector<tuple<ll, int, int>> ret;
        for (int i = 1; i < len(vs); i++) {
            int v = vs[i];
            while (len(st) >= 2 && lca.lca(v, st.back()) != st.back()) {
                int a = end(st)[-2];
                int b = st.back();
                ll c = lca.dist(a, b);
                ret.eb(c, a, b);
                st.pop_back();
            }
            st.pb(v);
        }
        while (len(st) >= 2) {
            int a = end(st)[-2];
            int b = st.back();
            ll c = lca.dist(a, b);
            ret.eb(c, a, b);
            st.pop_back();
        }
        return {ret, st.back()};
    }
};
```

9 Linear Algebra

9.1 Matrix (primitive)

```cpp
#include "../Contest/template.cpp"
template <typename T>
struct Matrix {
    int n, m;
    valarray<valarray<T>> v;
    Matrix(int _n, int _m, int id = 0) : n(_n), m(_m), v(valarray<T>(m), n
) {
        if (id) {
            rep(i, 0, n) v[i][i] = 1;
        }
    }
    valarray<T> &operator[](int x) { return v[x]; }
    Matrix transpose() {
        Matrix newv(m, n);
        rep(i, 0, n) rep(j, 0, m) newv[j][i] = (*this)[i][j];
```

```
    return newv;
}
Matrix operator+(Matrix &b) {
    Matrix ret(*this);
    return ret.v += b.v;
}
Matrix &operator+=(Matrix &b) { return v += b.v; }
Matrix operator*(Matrix b) {
    Matrix ret(n, b.m);
    rep(i, 0, n) rep(j, 0, m) rep(k, 0, b.m) ret[i][k] +=
        v[i][j] * b.v[j][k];
    return ret;
}
Matrix &operator*=(Matrix b) { return *this = *this * b; }
Matrix power(ll exp) {
    Matrix in = *this;
    Matrix ret(n, n, 1);
    while (exp) {
        if (exp & 1) ret *= in;
        in *= in;
        exp >>= 1;
    }
    return ret;
}
/*
 * Alters current matrix.
 * Does gaussian elimination and puts matrix in
 * upper echelon form (possibly reduced).
 * Returns the determinant of the square matrix
 * with side equal to the number of rows of the
 * original matrix.
 */
T gaussjordanize(int reduced = 0) {
    T det = T(1);
    int line = 0;
    rep(col, 0, m) {
        int pivot = line;
        while (pivot < n && v[pivot][col] == T(0)) pivot++;
        if (pivot >= n) continue;
        swap(v[line], v[pivot]);
        if (line != pivot) det *= T(-1);
        det *= v[line][line];
        v[line] /= T(v[line][col]);
        if (reduced) rep(i, 0, line) {
                v[i] -= T(v[i][col]) * v[line];
            }
        rep(i, line + 1, n) { v[i] -= T(v[i][col]) * v[line]; }
        line++;
    }
    return det * (line == n);
}
/*
 * Needs to be called in a square matrix that
 * represents a system of linear equations. Returns {possible solution
,
 * number of solutions (2 if infinite solutions)}
 */
pair<vector<T>, int> solve_system(vector<T> results) {
    Matrix aux(n, m + 1);
    rep(i, 0, n) {
        rep(j, 0, m) aux[i][j] = v[i][j];
        aux[i][m] = results[i];
    }
    T det = aux.gaussjordanize(1);
    int ret = 1 + (det == T(0));
    int n = results.size();
    rrep(i, n - 1, 0 - 1) {
        int pivot = 0;
        while (pivot < n && aux[i][pivot] == T(0)) pivot++;
        if (pivot == n) {
            if (aux[i][m] != T(0)) ret = 0;
        } else
            swap(aux[i], aux[pivot]);
    }
    rrep(i, n - 1, 0 - 1) rep(j, i + 1, n) aux[i][m] -=
        aux[i][j] * aux[j][m];
    rep(i, 0, n) results[i] = aux[i][m];
    rep(i, 0, n) rep(j, 0, m) v[i][j] = aux[i][j];
    return {results, ret};
}
/* Does not alter current matrix. Returns {inverse matrix, is curent
 * matrix invertable} */
pair<Matrix<T>, bool> find_inverse() {
    int n = v.size();
    Matrix<T> aug(n, 2 * n);
    rep(i, 0, n) rep(j, 0, n) aug[i][j] = v[i][j];
    rep(i, 0, n) aug[i][n + i] = 1;
    T det = aug.gaussjordanize(1);
    Matrix<T> ret(n, n);
    rep(i, 0, n) ret[i] = valarray<T>(aug[i][slice(n, n, 1)]);
    return {ret, det != T(0)};
}
/* Returns rank of matrix. does not alter it. */
int get_rank() const {
    if (m == 0) return 0;
    Matrix<T> aux(*this);
    aux.gaussjordanize();
    int resp = 0;
    rep(i, 0, n) resp += (aux[i] != valarray<T>(m)).sum();
    return resp;
```

```
    }
};
```

10 Math

10.1 Arithmetic Progression Sum

Usage:

- s : first term
- d : common difference
- n : number of terms

```
ll arithmeticProgressionSum(ll s, ll d, ll n) {
    return (s + (s + d * (n - 1))) * n / 2ll;
}
```

10.2 Binomial

Time: $O(N \cdot K)$
Memory: $O(K)$

```
ll binom(ll n, ll k) {
    if (k > n) return 0;
    vll dp(k + 1, 0);
    dp[0] = 1;
    for (ll i = 1; i <= n; i++)
        for (ll j = k; j > 0; j--) dp[j] = dp[j] + dp[j - 1];
    return dp[k];
}
```

10.3 Binomial MOD

Description: find $\binom{n}{k} \pmod{MOD}$
Time:

- precompute: on first call it takes $O(MAXNBIN)$ to precompute the factorials
- query: $O(1)$.

Memory: $O(MAXNBIN)$
Warning: Remember to set $MAXNBIN$ properly !

```
const ll MOD = 998244353;
inline ll binom(ll n, ll k) {
    static const int BINMAX = 2'000'000;
    static vll FAC(BINMAX + 1), FINV(BINMAX + 1);
    static bool done = false;
    if (!done) {
        vll INV(BINMAX + 1);
        FAC[0] = FAC[1] = INV[1] = FINV[0] = FINV[1] = 1;
        for (int i = 2; i <= BINMAX; i++) {
            FAC[i] = FAC[i - 1] * i % MOD;
            INV[i] = MOD - MOD / i * INV[MOD % i] % MOD;
            FINV[i] = FINV[i - 1] * INV[i] % MOD;
        }
        done = true;
    }
    if (n < k || n < 0 || k < 0) return 0;
    return FAC[n] * FINV[k] % MOD * FINV[n - k] % MOD;
}
```

10.4 Chinese Remainder Theorem

Description: Find the solution X to the N modular equations.

$$\begin{aligned} x &\equiv a_1 (mod m_1) \\ &\cdots \\ x &\equiv a_n (mod m_n) \end{aligned} \tag{1}$$

The m_i don't need to be coprime, if there is no solution then it returns -1.
Warning: Make sure all M_i are coprime !

```
#include "../Contest/template.cpp"
tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b % a, a);
    return {g, y - b / a * x, x};
}
template <typename T = ll>
struct crt {
    T a, m;
    crt() : a(0), m(1) {}
    crt(T a_, T m_) : a(a_), m(m_) {}
    crt operator*(crt C) {
        auto [g, x, y] = ext_gcd(m, C.m);
        if ((a - C.a) % g != 0) a = -1;
        if (a == -1 or C.a == -1) return crt(-1, 0);
        T lcm = m / g * C.m;
        T ans = a + (x * (C.a - a) / g % (C.m / g)) * m;
        return crt((ans % lcm + lcm) % lcm, lcm);
    }
};
template <typename T = ll>
struct Congruence {
    T a, m;
};
template <typename T = ll>
T chinese_remainder_theorem(const vector<Congruence<T>> &equations) {
    crt<T> ans;
    for (auto &[a_, m_] : equations) {
        ans = ans * crt<T>(a_, m_);
    }
    return ans.a;
}
```

10.5 Derangement / Matching Problem

Description: Computes the derangement of N, which is given by the formula: $D_N = N!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + ... + (-1)^N \frac{1}{N!}\right)$
Time: $O(N)$

```
#warning Remember to call precompute !
const ll MOD = 1e9 + 7;
const int MAXN(1 '000' 000);
ll fats[MAXN + 1];
void precompute() {
    fats[0] = 1;
    for (ll i = 1; i <= MAXN; i++) {
        fats[i] = (fats[i - 1] * i) % MOD;
    }
}
ll fastpow(ll a, ll p, ll m) {
    ll ret = 1;
    while (p) {
        if (p & 1) ret = (ret * a) % MOD;
        p >>= 1;
        a = (a * a) % MOD;
    }
    return ret;
}
ll divmod(ll a, ll b) { return (a * fastpow(b, MOD - 2, MOD)) % MOD; }
ll derangement(const ll n) {
    ll ans = fats[n];
    for (ll i = 1; i <= n; i++) {
        ll k = divmod(fats[n], fats[i]);
        if (i & 1) {
            ans = (ans - k + MOD) % MOD;
        } else {
            ans = (ans + k) % MOD;
        }
    }
    return ans;
}
```

10.6 Euler Phi

Description: Computes the number of positive integers less than N that are coprimes with N, in $O(\sqrt{N})$.

```
int phi(int n) {
    if (n == 1) return 1;
    auto fs = factorization(n);  // a vctor of pair or a map
    auto res = n;
    for (auto [p, k] : fs) {
        res /= p;
        res *= (p - 1);
    }
    return res;
}
```

10.7 Euler phi (in range)

Description: Computes the number of positive integers less than n that are coprimes with N, in the range $[1, N]$, in $O(N \log N)$.

```
const int MAX = 1e6;
vi range_phi(int n) {
    bitset<MAX> sieve;
    vi phi(n + 1);
    iota(phi.begin(), phi.end(), 0);
    sieve.set();
    for (int p = 2; p <= n; p += 2) phi[p] /= 2;
    for (int p = 3; p <= n; p += 2) {
        if (sieve[p]) {
            for (int j = p; j <= n; j += p) {
                sieve[j] = false;
                phi[j] /= p;
                phi[j] *= (p - 1);
            }
        }
    }
    return phi;
}
```

10.8 Extended Euclidian algorithm

Description: Finds the gcd between a and b and x and y such that $ax + by = g$
Time: $O(\log min(a, b))$
Warning: If $a = b = 0$ then there is infity solutions, but 0 is returned. Be careful about overflow.

```
#pragma once
#include "../Contest/template.cpp"
template <typename T>
tuple<T, T, T> extGcd(T a, T b) {
    if (!b) return {a, 1, 0};
    auto [d, x1, y1] = extGcd(b, a % b);
    T x = y1, y = x1 - y1 * (a / b);
    return {d, x, y};
}
```

10.9 FFT convolution and exponentiation

```
const ld PI = acos(-1);
/* change the ld to doulbe may increase
 * performance =D */
struct num {
    ld a{0.0}, b{0.0};
    num() {}
    num(ld na) : a{na} {}
```

```
    num(ld na, ld nb) : a{na}, b{nb} {}
    const num operator+(const num &c) const { return num(a + c.a, b + c.b)
    ; }
    const num operator-(const num &c) const { return num(a - c.a, b - c.b)
    ; }
    const num operator*(const num &c) const {
        return num(a * c.a - b * c.b, a * c.b + b * c.a);
    }
    const num operator/(const ll &c) const { return num(a / c, b / c); }
};
void fft(vector<num> &a, bool invert) {
    int n = len(a);
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1) j ^= bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    for (int sz = 2; sz <= n; sz <<= 1) {
        ld ang = 2 * PI / sz * (invert ? -1 : 1);
        num wsz(cos(ang), sin(ang));
        for (int i = 0; i < n; i += sz) {
            num w(1);
            rep(j, 0, sz / 2) {
                num u = a[i + j], v = a[i + j + sz / 2] * w;
                a[i + j] = u + v;
                a[i + j + sz / 2] = u - v;
                w = w * wsz;
            }
        }
    }
    if (invert)
        for (num &x : a) x = x / n;
}
vi conv(vi const a, vi const b) {
    vector<num> fa(all(a));
    vector<num> fb(all(b));
    int n = 1;
    while (n < len(a) + len(b)) n <<= 1;
    fa.resize(n);
    fb.resize(n);
    fft(fa, false);
    fft(fb, false);
    rep(i, 0, n) fa[i] = fa[i] * fb[i];
    fft(fa, true);
    vi result(n);
    rep(i, 0, n) result[i] = round(fa[i].a);
    while (len(result) and result.back() == 0) result.pop_back();

    /* Unconment this line if you want a boolean
     * convolution*/
    // for (auto &xi : result) xi = min(xi, 1ll);

    return result;
}
vll poly_exp(vll &ps, int k) {
    vll ret(len(ps));
    auto base = ps;
    ret[0] = 1;
    while (k) {
        if (k & 1) ret = conv(ret, base);
        k >>= 1;
        base = conv(base, base);
    }
    return ret;
}
```

10.10 Factorial Factorization

Description: Computes the factorization of $N!$ in $\varphi(N) * \log N$
Time: $O(\varphi(N) \cdot \log N)$

```
ll E(ll n, ll p) {
    ll k = 0, b = p;
    while (b <= n) {
        k += n / b;
        b *= p;
    }
    return k;
}
map<ll, ll> factorial_factorization(ll n, const vll &primes) {
    map<ll, ll> fs;
    for (const auto &p : primes) {
        if (p > n) break;
        fs[p] = E(n, p);
    }
    return fs;
}
```

10.11 Factorization

Description: Computes the factorization of N.
Time: $O(\sqrt{n})$.

```
map<ll, ll> factorization(ll n) {
    map<ll, ll> ans;
    for (ll i = 2; i * i <= n; i++) {
        ll count = 0;
        for (; n % i == 0; count++, n /= i);
        if (count) ans[i] = count;
    }
    if (n > 1) ans[n]++;
    return ans;
}
```

10.12 Factorization (Pollard's Rho)

Description: Factorizes a number into its prime factors.
Time: $O(N^{(\frac{1}{4})} * \log(N))$.

```
ll mul(ll a, ll b, ll m) {
    ll ret = a * b - (ll)((ld)1 / m * a * b + 0.5) * m;
    return ret < 0 ? ret + m : ret;
}
ll pow(ll a, ll b, ll m) {
    ll ans = 1;
    for (; b > 0; b /= 2ll, a = mul(a, a, m)) {
        if (b % 2ll == 1) ans = mul(ans, a, m);
    }
    return ans;
}
bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;
    ll r = __builtin_ctzll(n - 1), d = n >> r;
    for (int a : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        ll x = pow(a, d, n);
        if (x == 1 or x == n - 1 or a % n == 0) continue;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) break;
        }
        if (x != n - 1) return 0;
    }
    return 1;
}
ll rho(ll n) {
    if (n == 1 or prime(n)) return n;
    auto f = [n](ll x) { return mul(x, x, n) + 1; };
    ll x = 0, y = 0, t = 30, prd = 2, x0 = 1, q;
    while (t % 40 != 0 or gcd(prd, n) == 1) {
        if (x == y) x = ++x0, y = f(x);
        q = mul(prd, abs(x - y), n);
        if (q != 0) prd = q;
        x = f(x), y = f(f(y)), t++;
    }
    return gcd(prd, n);
}
vector<ll> fact(ll n) {
    if (n == 1) return {};
    if (prime(n)) return {n};
    ll d = rho(n);
    vector<ll> l = fact(d), r = fact(n / d);
    l.insert(l.end(), r.begin(), r.end());
    return l;
}
```

10.13 Fast Pow

Description: Computes $a^b \pmod m$
Time: $O(\log B)$.

```
ll fpow(ll a, ll b, ll m) {
    ll ret = 1;
    while (b) {
        if (b & 1) ret = (ret * a) % m;
        b >>= 1;
        a = (a * a) % m;
    }
    return ret;
}
```

10.14 Find linear recurrence (Berlekamp-Massey)

Description: Given the first N terms of a linear recurrence finds the smallest recurrence that matches the sequence.
Time: $O(N^2)$
Warning: Works faster if the *mod* is const but can be also be a parameter.
Absolute magic !

```
const ll mod = 998244353;
ll modpow(ll b, ll e) {
    ll ans = 1;
    for (; e; b = b * b % mod, e /= 2)
        if (e & 1) ans = ans * b % mod;
    return ans;
}
vl berlekampMassey(vll s) {
    int n = len(s), L = 0, m = 0;
    if (!n) return {};
    vll C(n), B(n), T;
    C[0] = B[0] = 1;
    ll b = 1;
    rep(i, 0, n) {
        ++m;
        ll d = s[i] % mod;
        rep(j, 1, L + 1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C;
        ll coef = d * modpow(b, mod - 2) % mod;
        rep(j, m, n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L;
        B = T;
        b = d;
        m = 0;
    }
    C.resize(L + 1);
    C.erase(C.begin());
    for (ll &x : C) x = (mod - x) % mod;
    return C;
}
```

10.15 Find multiplicatinve inverse

```
ll inv(ll a, ll m) { return a > 1ll ? m - inv(m % a, a) * m / a : 1ll; }
```

10.16 Floor division

```
template <typename T1, typename T2>
constexpr typename std::common_type<T1, T2>::type floor_div(T1 x, T2 y) {
    assert(y != 0);
    if (y < 0) x = -x, y = -y;
    return x < 0 ? (x - y + 1) / y : x / y;
}
```

10.17 GCD

```
template <typename T>
T gcd(T a, T b) {
    return b ? gcd(b, a % b) : a;
}
```

10.18 Gauss XOR elimination / XOR-SAT

Description: Execute gaussian elimination with xor over the system $Ax = b$ in. The add method must receive a bitset indicating which variables are present in the equation, and the solution of the equation.

Time: $O(\frac{nm^2}{64})$

```
const int MAXXI = 2009;
using Equation = bitset<MAXXI>;
struct GaussXor {
    vector<char> B;
    vector<Equation> A;
    void add(const Equation &ai, bool bi) {
        A.push_back(ai);
        B.push_back(bi);
    }
    pair<bool, Equation> solution() {
        int cnt = 0, n = A.size();
        Equation vis;
        vis.set();
        Equation x;
        for (int j = MAXXI - 1, i; j >= 0; j--) {
            for (i = cnt; i < n; i++) {
                if (A[i][j]) break;
            }
            if (i == n) continue;
            swap(A[i], A[cnt]), swap(B[i], B[cnt]);
            i = cnt++;
            vis[j] = 0;
            for (int k = 0; k < n; k++) {
                if (i == k || !A[k][j]) continue;
                A[k] ^= A[i];
                B[k] ^= B[i];
            }
        }
        x = vis;
        for (int i = 0; i < n; i++) {
            int acum = 0;
            for (int j = 0; j < MAXXI; j++) {
                if (!A[i][j]) continue;
                if (!vis[j]) {
                    vis[j] = 1;
                    x[j] = acum ^ B[i];
                }
                acum ^= x[j];
            }
            if (acum != B[i]) return {false, Equation()};
        }
        return {true, x};
    }
};
```

10.19 Guess K-th (Berlekamp-Massey)

```
/* Berlekamp-Massey algorithm
 *
 * Given the first n terms of a linear recurrence relation, this algorithm
 * finds the shortest linear recurrence relation that generates the given
 * sequence.
 *
 * Note: mod needs to have inverse
 *
 * Time complexity: O(n^2)
 */
template <typename T>
vector<T> berlekamp_massey(const vector<T> &s) {
    vector<T> cur, best;
    int lf, ld;
    for (int i = 0; i < (int)s.size(); i++) {
        T delta = 0;
        for (int j = 0; j < (int)cur.size(); j++)
            delta += s[i - j - 1] * cur[j];
        if (delta == s[i]) continue;
        if (cur.empty()) {
            cur.resize(i + 1);
            lf = i;
            ld = (int)(delta - s[i]).value();
            continue;
        }
        T coef = -(s[i] - delta) / ld;
        vector<T> c(i - lf - 1);
        c.push_back(coef);
        for (auto &x : best) c.push_back(-x * coef);
        if (c.size() < cur.size()) c.resize(cur.size());
        for (int j = 0; j < (int)cur.size(); j++) c[j] += cur[j];
        if (i - lf + (int)best.size() >= (int)cur.size())
            best = cur, lf = i, ld = (int)(delta - s[i]).value();
        cur = c;
    }
    return cur;
}
template <typename T>
T get_kth(const vector<T> &rec, const vector<T> &dp, ll k) {
```

```
    int n = (int)rec.size();
    assert(rec.size() <= dp.size());
    // use fft to speed up
    auto mul = [&](const vector<T> &a, const vector<T> &b) {
        vector<T> res(2 * n);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) res[i + j] += a[i] * b[j];
        for (int i = 2 * n - 1; i >= n; i--)
            for (int j = 1; j <= n; j++) res[i - j] += res[i] * rec[j -
   1];
        res.resize(n);
        return res;
    };
    vector<T> a(n), x(n);
    x[0] = 1;
    if (n != 1)
        a[1] = 1;
    else
        a[0] = rec[0];
    while (k) {
        if (k & 1) x = mul(x, a);
        a = mul(a, a);
        k >>= 1;
    }
    T res = 0;
    for (int i = 0; i < n; i++) res += x[i] * dp[i];
    return res;
}
template <typename T>
T guess_kth_term(const vector<T> &s, ll k) {
    if (k < (int)s.size()) return s[k];
    auto coef = berlekamp_massey(s);
    if (coef.empty()) return 0;
    return get_kth(coef, s, k);
}
```

10.20 Integer partition

Description: Find the total of ways to partition a given number N in such way that none of the parts is greater than K.
Time: $O(N \cdot min(N, K))$
Memory: $O(N)$
Warning: Remember to memset everything to -1 before using it

```
const ll MOD = 1000000007;
const int MAXN(100);
ll memo[MAXN + 1];
ll dp(ll n, ll k = oo) {
    if (n == 0) return 1;
    ll &ans = memo[n];
    if (ans != -1) return ans;
    ans = 0;
    for (int i = 1; i <= min(n, k); i++) {
        ans = (ans + dp(n - i, k)) % MOD;
    }
    return ans;
}
```

10.21 LCM

```
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : a; }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
```

10.22 Linear Recurrence

Description: Find the n-th term of a linear recurrence, given the recurrence *rec* and the first K values of the recurrence, remember that first_k[i] is the value of $f(i)$, considering 0-indexing.
Usage: Suppose you want the N-th term of Fibonacci the first k should be 1,1, and the rec should be 0,1,1,1.
Time: $O(K^3 \log N)$

```
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a, vector<vector<T>> &b,
                       const ll mod) {
    assert(a.back().size() == b.size());
    int n = a.size();
    int m = a.back().size();
    vector<vector<T>> c(n, vector<T>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < m; k++) {
                c[i][j] = (c[i][j] + ((a[i][k] * b[k][j]) % mod)) % mod;
            }
        }
    }
    return c;
}
template <typename T>
vector<vector<T>> fpow(vector<vector<T>> &xs, ll p, ll mod) {
    vector<vector<T>> ans(xs.size(), vector<T>(xs.size()));
    for (int i = 0; i < (int)xs.size(); i++) ans[i][i] = 1;
    for (auto b = xs; p; p >>= 1, b = prod(b, b, mod))
        if (p & 1) ans = prod(ans, b, mod);
    return ans;
}
ll linear_req(vector<vector<ll>> rec, vector<ll> first_k, ll n, const ll
   mod) {
    int k = first_k.size();
    if (n <= k) return first_k[n - 1];
    ll n2 = n - k;
    rec = fpow(rec, n2, mod);
    ll ret = 0;
    for (int i = 0; i < k; i++)
        ret = (ret + (rec.back()[i] * first_k[i]) % mod) % mod;
    return ret;
}
```

10.23 Linear diophantine equation (count)

Description:
Time: $O(\log min(a,b))$

```
#pragma once
#include "../Contest/template.cpp"
#include "./Extended Euclidian algorithm.cpp"
#include "./Linear diophantine equation (solve).cpp"

template <typename T>
T countSolutionsInRange(T a, T b, T c, T minX, T maxX, T minY, T maxY) {
    auto ss = [&](T &x, T &y, T a, T b, T cnt) { x += cnt * b, y -= cnt *
    a; };
    assert(a and b);

    auto sol = diophantineEquationSolution(a, b, c);
    if (!sol) return 0;

    auto [x, y] = *sol;
    auto g = get<0>(extGcd(a, b));
    a /= g;
    b /= g;

    int signA = a > 0 ? +1 : -1;
    int signB = b > 0 ? +1 : -1;

    ss(x, y, a, b, (minX - x) / b);
    if (x < minX) ss(x, y, a, b, signB);
    if (x > maxX) return 0;
    int lx1 = x;

    ss(x, y, a, b, (maxX - x) / b);
    if (x > maxX) ss(x, y, a, b, -signB);
    int rx1 = x;

    ss(x, y, a, b, -(minY - y) / a);
    if (y < minY) ss(x, y, a, b, -signA);
    if (y > maxY) return 0;
    int lx2 = x;

    ss(x, y, a, b, -(maxY - y) / a);
    if (y > maxY) ss(x, y, a, b, signA);
    int rx2 = x;

    if (lx2 > rx2) swap(lx2, rx2);
    int lx = max(lx1, lx2);
    int rx = min(rx1, rx2);

    if (lx > rx) return 0;
    return (rx - lx) / abs(b) + 1;
}
```

10.24 Linear diophantine equation (solve)

Description: Finds a solution for $ax + by = c$, where a, b, c, are given and x and y unknown.
Time: $O(\log min(a,b))$

```
#pragma once
#include "../Contest/template.cpp"
#include "./Extended Euclidian algorithm.cpp"

template <typename T>
optional<pair<T, T>> diophantineEquationSolution(T a, T b, T c) {
    if (a == 0 and b == 0) {
        if (c)
            return nullopt;
        else
            return pair<T, T>{(T)0, (T)0};
    }
    auto [g, x0, y0] = extGcd(a < 0 ? a * -1 : a, b < 0 ? b * -1 : b);

    if (c % g) return nullopt;

    x0 *= c / g, y0 *= c / g;

    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;

    pair<T, T> ret;
    ret.first = x0, ret.second = y0;
    return ret;
}
```

10.25 List N elements choose K

Description: Process every possible combination of K elements from N elements, thoose index marked as 1 in the index vector says which elments are choosed at that moment.
Time: $O(\binom{N}{K} \cdot O(process))$

```
void process(vi &index) {
    for (int i = 0; i < len(index); i++) {
        if (index[i]) cout << i << " \n"[i == len(index) - 1];
    }
}
void n_choose_k(int n, in k) {
    vi index(n);
    fill(index.end() - k, index.end(), 1);

    do {
        process(index);
    } while (next_permutation(all(index)));
}
```

10.26 List primes (Sieve of Eratosthenes)

```
const ll MAXN = 2e5;
vll list_primes(ll n = MAXN) {
    vll ps;
    bitset<MAXN + 1> sieve;
    sieve.set();
    sieve.reset(1);
    for (ll i = 2; i <= n; ++i) {
        if (sieve[i]) ps.push_back(i);
        for (ll j = i * 2; j <= n; j += i) {
            sieve.reset(j);
        }
    }
```

```
    return ps;
}
```

10.27 Matrix exponentiation

```
const ll MOD = 1 '000' 000'007;
template <typename T>
vector<vector<T>> prod(vector<vector<T>> &a, vector<vector<T>> &b) {
    int n = len(a);
    vector<vector<T>> c(n, vector<T>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                c[i][j] = (c[i][j] + ((a[i][k] * b[k][j]) % MOD)) % MOD;
            }
        }
    }
    return c;
}
template <typename T>
vector<vector<T>> fpow(vector<vector<T>> &xs, ll p) {
    vector<vector<T>> ans(len(xs), vector<T>(len(xs)));
    for (int i = 0; i < len(xs); i++) ans[i][i] = 1;
    auto b = xs;
    while (p) {
        if (p & 1) ans = prod(ans, b);
        p >>= 1;
        b = prod(b, b);
    }
    return ans;
}
```

10.28 NTT integer convolution and exponentiation

Time:
- Convolution $O(N \cdot \log N)$,
- Exponentiation: $O(\log K \cdot N \cdot \log N)$

```
template <int _mod>
struct mint {
    ll expo(ll b, ll e) {
        ll ret = 1;
        while (e) {
            if (e % 2) ret = ret * b % _mod;
            e /= 2, b = b * b % _mod;
        }
        return ret;
    }
    ll inv(ll b) { return expo(b, _mod - 2); }
    using m = mint;
    ll v;
    mint() : v(0) {}
    mint(ll v_) {
        if (v_ >= _mod or v_ <= -_mod) v_ %= _mod;
        if (v_ < 0) v_ += _mod;
        v = v_;
    }
    m &operator+=(const m &a) {
        v += a.v;
        if (v >= _mod) v -= _mod;
        return *this;
    }
    m &operator-=(const m &a) {
        v -= a.v;
        if (v < 0) v += _mod;
        return *this;
    }
    m &operator*=(const m &a) {
        v = v * ll(a.v) % _mod;
        return *this;
    }
    m &operator/=(const m &a) {
        v = v * inv(a.v) % _mod;
        return *this;
    }
    m operator-() { return m(-v); }
    m &operator^=(ll e) {
        if (e < 0) {
            v = inv(v);
            e = -e;
        }
        v = expo(v, e);
        // possivel otimizacao:
        // cuidado com 0^0
        // v = expo(v, e%(p-1));
        return *this;
    }
    bool operator==(const m &a) { return v == a.v; }
    bool operator!=(const m &a) { return v != a.v; }
    friend istream &operator>>(istream &in, m &a) {
        ll val;
        in >> val;
        a = m(val);
        return in;
    }
    friend ostream &operator<<(ostream &out, m a) { return out << a.v; }
    friend m operator+(m a, m b) { return a += b; }
    friend m operator-(m a, m b) { return a -= b; }
    friend m operator*(m a, m b) { return a *= b; }
    friend m operator/(m a, m b) { return a /= b; }
    friend m operator^(m a, ll e) { return a ^= e; }
};
const ll MOD1 = 998244353;
const ll MOD2 = 754974721;
const ll MOD3 = 167772161;
template <int _mod>
void ntt(vector<mint<_mod>> &a, bool rev) {
    int n = len(a);
```

```cpp
    auto b = a;
    assert(!(n & (n - 1)));
    mint<_mod> g = 1;
    while ((g ^ (_mod / 2)) == 1) g += 1;
    if (rev) g = 1 / g;
    for (int step = n / 2; step; step /= 2) {
        mint<_mod> w = g ^ (_mod / (n / step)), wn = 1;
        for (int i = 0; i < n / 2; i += step) {
            for (int j = 0; j < step; j++) {
                auto u = a[2 * i + j], v = wn * a[2 * i + j + step];
                b[i + j] = u + v;
                b[i + n / 2 + j] = u - v;
            }
            wn = wn * w;
        }
        swap(a, b);
    }
    if (rev) {
        auto n1 = mint<_mod>(1) / n;
        for (auto &x : a) x *= n1;
    }
}
template <ll _mod>
vector<mint<_mod>> convolution(const vector<mint<_mod>> &a,
                               const vector<mint<_mod>> &b) {
    vector<mint<_mod>> l(all(a)), r(all(b));
    int N = len(l) + len(r) - 1, n = 1;
    while (n <= N) n *= 2;
    l.resize(n), r.resize(n);
    ntt(l, false), ntt(r, false);
    for (int i = 0; i < n; i++) l[i] *= r[i];
    ntt(l, true);
    l.resize(N);
    // Uncommnent for a boolean convolution :)
    /*
    for (auto& li : l) {
      li.v = min(li.v, 1ll);
    }
    */
    return l;
}
template <ll _mod>
vector<mint<_mod>> poly_exp(vector<mint<_mod>> &ps, int k) {
    vector<mint<_mod>> ret(len(ps));
    auto base = ps;
    ret[0] = 1;
    while (k) {
        if (k & 1) ret = convolution(ret, base);
        k >>= 1;
        base = convolution(base, base);
    }
    return ret;
}
```

10.29 NTT integer convolution and exponentiation (2 mods modules)

Description: Computes the convolution between the two polynomials and.
Time: $O(N \log N)$
Warning: This is pure magic !

```cpp
template <int _mod>
struct mint {
    ll expo(ll b, ll e) {
        ll ret = 1;
        while (e) {
            if (e % 2) ret = ret * b % _mod;
            e /= 2, b = b * b % _mod;
        }
        return ret;
    }
    ll inv(ll b) { return expo(b, _mod - 2); }
    using m = mint;
    ll v;
    mint() : v(0) {}
    mint(ll v_) {
        if (v_ >= _mod or v_ <= -_mod) v_ %= _mod;
        if (v_ < 0) v_ += _mod;
        v = v_;
    }
    m &operator+=(const m &a) {
        v += a.v;
        if (v >= _mod) v -= _mod;
        return *this;
    }
    m &operator-=(const m &a) {
        v -= a.v;
        if (v < 0) v += _mod;
        return *this;
    }
    m &operator*=(const m &a) {
        v = v * ll(a.v) % _mod;
        return *this;
    }
    m &operator/=(const m &a) {
        v = v * inv(a.v) % _mod;
        return *this;
    }
    m operator-() { return m(-v); }
    m &operator^=(ll e) {
        if (e < 0) {
            v = inv(v);
            e = -e;
        }
        v = expo(v, e);
        // possivel otimizacao:
        // cuidado com 0^0
        // v = expo(v, e%(p-1));
        return *this;
    }
```

```cpp
    bool operator==(const m &a) { return v == a.v; }
    bool operator!=(const m &a) { return v != a.v; }
    friend istream &operator>>(istream &in, m &a) {
        ll val;
        in >> val;
        a = m(val);
        return in;
    }
    friend ostream &operator<<(ostream &out, m a) { return out << a.v; }
    friend m operator+(m a, m b) { return a += b; }
    friend m operator-(m a, m b) { return a -= b; }
    friend m operator*(m a, m b) { return a *= b; }
    friend m operator/(m a, m b) { return a /= b; }
    friend m operator^(m a, ll e) { return a ^= e; }
};
const ll MOD1 = 998244353;
const ll MOD2 = 754974721;
const ll MOD3 = 167772161;
template <int _mod>
void ntt(vector<mint<_mod>> &a, bool rev) {
    int n = len(a);
    auto b = a;
    assert(!(n & (n - 1)));
    mint<_mod> g = 1;
    while ((g ^ (_mod / 2)) == 1) g += 1;
    if (rev) g = 1 / g;
    for (int step = n / 2; step; step /= 2) {
        mint<_mod> w = g ^ (_mod / (n / step)), wn = 1;
        for (int i = 0; i < n / 2; i += step) {
            for (int j = 0; j < step; j++) {
                auto u = a[2 * i + j], v = wn * a[2 * i + j + step];
                b[i + j] = u + v;
                b[i + n / 2 + j] = u - v;
            }
            wn = wn * w;
        }
        swap(a, b);
    }
    if (rev) {
        auto n1 = mint<_mod>(1) / n;
        for (auto &x : a) x *= n1;
    }
}
tuple<ll, ll, ll> ext_gcd(ll a, ll b) {
    if (!a) return {b, 0, 1};
    auto [g, x, y] = ext_gcd(b % a, a);
    return {g, y - b / a * x, x};
}
template <typename T = ll>
struct crt {
    T a, m;
    crt() : a(0), m(1) {}
    crt(T a_, T m_) : a(a_), m(m_) {}
    crt operator*(crt C) {
        auto [g, x, y] = ext_gcd(m, C.m);
        if ((a - C.a) % g != 0) a = -1;
        if (a == -1 or C.a == -1) return crt(-1, 0);
        T lcm = m / g * C.m;
        T ans = a + (x * (C.a - a) / g % (C.m / g)) * m;
        return crt((ans % lcm + lcm) % lcm, lcm);
    }
};
template <typename T = ll>
struct Congruence {
    T a, m;
};
template <typename T = ll>
T chinese_remainder_theorem(const vector<Congruence<T>> &equations) {
    crt<T> ans;
    for (auto &[a_, m_] : equations) {
        ans = ans * crt<T>(a_, m_);
    }
    return ans.a;
}
#define int long long
template <ll m1, ll m2>
vll merge_two_mods(const vector<mint<m1>> &a, const vector<mint<m2>> &b) {
    int n = len(a);
    vll ans(n);
    for (int i = 0; i < n; i++) {
        auto cur = crt<ll>();
        auto ai = a[i].v;
        auto bi = b[i].v;
        cur = cur * crt<ll>(ai, m1);
        cur = cur * crt<ll>(bi, m2);
        ans[i] = cur.a;
    }
    return ans;
}
vll convolution_2mods(const vll &a, const vll &b) {
    vector<mint<MOD1>> l(all(a)), r(all(b));
    int N = len(l) + len(r) - 1, n = 1;
    while (n <= N) n *= 2;
    l.resize(n), r.resize(n);
    ntt(l, false), ntt(r, false);
    for (int i = 0; i < n; i++) l[i] *= r[i];
    ntt(l, true);
    l.resize(N);
    vector<mint<MOD2>> l2(all(a)), r2(all(b));
    l2.resize(n), r2.resize(n);
    ntt(l2, false), ntt(r2, false);
    rep(i, 0, n) l2[i] *= r2[i];
    ntt(l2, true);
    l2.resize(N);
    return merge_two_mods(l, l2);
}
vll poly_exp(const vll &xs, ll k) {
    vll ret(len(xs));
```

```
    ret[0] = 1;
    auto base = xs;
    while (k) {
        if (k & 1) ret = convolution_2mods(ret, base);
        k >>= 1;
        base = convolution_2mods(base, base);
    }
    return ret;
}
```

10.30 Polynomial Taylor Shift

```
using C = complex<double>;
const ll mod = 998244353;
void fft(vector<C> &a) {
    int n = len(a), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2) {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, acos(-1.0L) / k);
        for (int i = k; i < 2 * k; i++)
            rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vector<int> rev(n);
    for (int i = 0; i < n; i++) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    for (int i = 0; i < n; i++)
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k)
            for (int j = 0; j < k; j++) {
                auto x = (double *)&rt[j + k], y = (double *)&a[i + j + k
    ];
                C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
                a[i + j + k] = a[i + j] - z;
                a[i + j] += z;
            }
    }
}
vector<double> conv(const vector<double> &a, const vector<double> &b) {
    if (a.empty() || b.empty()) return {};
    vector<double> res(len(a) + len(b) - 1);
    int L = 32 - __builtin_clz(len(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(a.begin(), a.end(), begin(in));
    for (int i = 0; i < len(b); i++) in[i].imag(b[i]);
    fft(in);
    for (C &x : in) x *= x;
    for (int i = 0; i < n; i++) {
        out[i] = in[-i & (n - 1)] - conj(in[i]);
    }
    fft(out);
    for (int i = 0; i < len(res); i++) {
        res[i] = imag(out[i]) / (4 * n);
    }
    return res;
}
template <ll M>
vector<ll> convMod(const vector<ll> &a, const vector<ll> &b) {
    if (a.empty() || b.empty()) return {};
    vector<ll> res(len(a) + len(b) + 1);
    int B = 32 - __builtin_clz(len(res)), n = 1 << B, cut = int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    for (int i = 0; i < len(a); i++) {
        L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    }
    for (int i = 0; i < len(b); i++) {
        R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    }
    fft(L), fft(R);
    for (int i = 0; i < n; i++) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    for (int i = 0; i < len(res); i++) {
        ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
        ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
ll fexp(ll b, ll e) {
    ll res = 1;
    while (e > 0) {
        if (e & 1) res = res * b % mod;
        b = b * b % mod;
        e >>= 1;
    }
    return res;
}
ll inv(ll n) { return fexp(n, mod - 2); }
vector<ll> shift(vector<ll> &a, ll v) {
    int n = len(a) - 1;
    vector<ll> f(n + 1), g(n + 1);
    vector<ll> i_fact(n + 1);
    f[0] = a[0];
    g[n] = 1;
    i_fact[0] = 1;
    ll fact = 1, potk = 1;
    for (int i = 1; i < n + 1; i++) {
        fact = fact * i % mod;
        f[i] = fact * a[i] % mod;
        potk = (potk * v % mod + mod) % mod;
        g[n - i] = ((potk * inv(fact)) % mod + mod) % mod;
        i_fact[i] = inv(fact);
    }
    auto p = convMod<mod>(f, g);
```

```
    vector<ll> res(n + 1);
    for (int i = 0; i < n + 1; i++) {
        res[i] = (p[i + n] * i_fact[i] % mod + mod) % mod;
    }
    return res;
}
```

10.31 Polyominoes

Usage: *buildPolyominoes*(x) creates every polyomino until size x, and put it in *polyominoes*$[x]$, access *polyomino.v* to find the vector of pairs representing the coordinates of each piece, considering that the polyomino was 'rooted' in coordinate $(0, 0)$.
Warning: note that when accessing *polyominoes*$[x]$ only the first x coordinates are valid.

```
#include "../Contest/template.cpp"
const int MAXP = 10;
using pii = pair<int, int>;
// This implementation considers the rotations as
// distinct
//              0, 10, 10+9, 10+9+8...
int pos[11] = {0, 10, 19, 27, 34, 40, 45, 49, 52, 54, 55};
struct Polyominoes {
    pii v[MAXP];
    ll id;
    int n;
    Polyominoes() {
        n = 1;
        v[0] = {0, 0};
        normalize();
    }
    pii &operator[](int i) { return v[i]; }
    bool add(int a, int b) {
        for (int i = 0; i < n; i++)
            if (v[i].first == a and v[i].second == b) return false;
        v[n++] = pii(a, b);
        normalize();
        return true;
    }
    void normalize() {
        int mnx = 100, mny = 100;
        for (int i = 0; i < n; i++)
            mnx = min(mnx, v[i].first), mny = min(mny, v[i].second);
        id = 0;
        for (int i = 0; i < n; i++) {
            v[i].first -= mnx, v[i].second -= mny;
            id |= (1LL << (pos[v[i].first] + v[i].second));
        }
    }
};
vector<Polyominoes> polyominoes[MAXP + 1];
void buildPolyominoes(int mxN = 10) {
    vector<pair<int, int>> dt({{1, 0}, {-1, 0}, {0, -1}, {0, 1}});
    for (int i = 0; i <= mxN; i++) polyominoes[i].clear();
    Polyominoes init;
    queue<Polyominoes> q;
    unordered_set<int64_t> used;
    q.push(init);
    used.insert(init.id);
    while (!q.empty()) {
        Polyominoes u = q.front();
        q.pop();
        polyominoes[u.n].push_back(u);
        if (u.n == mxN) continue;
        for (int i = 0; i < u.n; i++) {
            for (auto [dx, dy] : dt) {
                Polyominoes to = u;
                bool ok = to.add(to[i].first + dx, to[i].second + dy);
                if (ok and !used.count(to.id)) {
                    q.push(to);
                    used.insert(to.id);
                }
            }
        }
    }
}
```

11 Primitives

11.1 Bigint

```
const int maxn = 1e2 + 14, lg = 15;
const int base = 1000000000;
const int base_digits = 9;
struct bigint {
    vi a;
    int sign;
    int size() {
        if (a.empty()) return 0;
        int ans = (a.size() - 1) * base_digits;
        int ca = a.back();
        while (ca) ans++, ca /= 10;
        return ans;
    }
    bigint operator^(const bigint &v) {
        bigint ans = 1, a = *this, b = v;
        while (!b.isZero()) {
            if (b % 2) ans *= a;
            a *= a, b /= 2;
        }
        return ans;
    }
    string to_string() {
        stringstream ss;
        ss << *this;
        string s;
        ss >> s;
        return s;
    }
    int sumof() {
        string s = to_string();
```

```
    int ans = 0;
    for (auto c : s) ans += c - '0';
    return ans;
}
/*</arpa>*/
bigint() : sign(1) {}
bigint(long long v) { *this = v; }
bigint(const string &s) { read(s); }
void operator=(const bigint &v) {
    sign = v.sign;
    a = v.a;
}
void operator=(long long v) {
    sign = 1;
    a.clear();
    if (v < 0) sign = -1, v = -v;
    for (; v > 0; v = v / base) a.push_back(v % base);
}
bigint operator+(const bigint &v) const {
    if (sign == v.sign) {
        bigint res = v;
        for (int i = 0, carry = 0;
             i < (int)max(a.size(), v.a.size()) || carry; ++i) {
            if (i == (int)res.a.size()) res.a.push_back(0);
            res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
            carry = res.a[i] >= base;
            if (carry) res.a[i] -= base;
        }
        return res;
    }
    return *this - (-v);
}
bigint operator-(const bigint &v) const {
    if (sign == v.sign) {
        if (abs() >= v.abs()) {
            bigint res = *this;
            for (int i = 0, carry = 0; i < (int)v.a.size() || carry;
++i) {
                res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0)
;
                carry = res.a[i] < 0;
                if (carry) res.a[i] += base;
            }
            res.trim();
            return res;
        }
        return -(v - *this);
    }
    return *this + (-v);
}
void operator*=(int v) {
    if (v < 0) sign = -sign, v = -v;
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
            long long cur = a[i] * (long long)v + carry;
            carry = (int)(cur / base);
            a[i] = (int)(cur % base);
            // asm("divl %%ecx" : "=a"(carry),
            // "=d"(a[i]) : "A"(cur), "c"(base));
        }
        trim();
}
bigint operator*(int v) const {
    bigint res = *this;
    res *= v;
    return res;
}
void operator*=(long long v) {
    if (v < 0) sign = -sign, v = -v;
    if (v > base) {
        *this = *this * (v / base) * base + *this * (v % base);
        return;
    }
    for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
        if (i == (int)a.size()) a.push_back(0);
        long long cur = a[i] * (long long)v + carry;
        carry = (int)(cur / base);
        a[i] = (int)(cur % base);
        // asm("divl %%ecx" : "=a"(carry),
        // "=d"(a[i]) : "A"(cur), "c"(base));
    }
    trim();
}
bigint operator*(long long v) const {
    bigint res = *this;
    res *= v;
    return res;
}
friend pair<bigint, bigint> divmod(const bigint &a1, const bigint &b1)
 {
    int norm = base / (b1.a.back() + 1);
    bigint a = a1.abs() * norm;
    bigint b = b1.abs() * norm;
    bigint q, r;
    q.a.resize(a.a.size());
    for (int i = a.a.size() - 1; i >= 0; i--) {
        r *= base;
        r += a.a[i];
        int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
        int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() -
1];
        int d = ((long long)base * s1 + s2) / b.a.back();
        r -= b * d;
        while (r < 0) r += b, --d;
        q.a[i] = d;
    }
    q.sign = a1.sign * b1.sign;
    r.sign = a1.sign;
    q.trim();
```

```
        r.trim();
        return make_pair(q, r / norm);
    }
    bigint operator/(const bigint &v) const { return divmod(*this, v).
first; }
    bigint operator%(const bigint &v) const { return divmod(*this, v).
second; }
    void operator/=(int v) {
        if (v < 0) sign = -sign, v = -v;
        for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
            long long cur = a[i] + rem * (long long)base;
            a[i] = (int)(cur / v);
            rem = (int)(cur % v);
        }
        trim();
    }
    bigint operator/(int v) const {
        bigint res = *this;
        res /= v;
        return res;
    }
    int operator%(int v) const {
        if (v < 0) v = -v;
        int m = 0;
        for (int i = a.size() - 1; i >= 0; --i)
            m = (a[i] + m * (long long)base) % v;
        return m * sign;
    }
    void operator+=(const bigint &v) { *this = *this + v; }
    void operator-=(const bigint &v) { *this = *this - v; }
    void operator*=(const bigint &v) { *this = *this * v; }
    void operator/=(const bigint &v) { *this = *this / v; }
    bool operator<(const bigint &v) const {
        if (sign != v.sign) return sign < v.sign;
        if (a.size() != v.a.size())
            return a.size() * sign < v.a.size() * v.sign;
        for (int i = a.size() - 1; i >= 0; i--)
            if (a[i] != v.a[i]) return a[i] * sign < v.a[i] * sign;
        return false;
    }
    bool operator>(const bigint &v) const { return v < *this; }
    bool operator<=(const bigint &v) const { return !(v < *this); }
    bool operator>=(const bigint &v) const { return !(*this < v); }
    bool operator==(const bigint &v) const {
        return !(*this < v) && !(v < *this);
    }
    bool operator!=(const bigint &v) const { return *this < v || v < *this
; }
    void trim() {
        while (!a.empty() && !a.back()) a.pop_back();
        if (a.empty()) sign = 1;
    }
    bool isZero() const { return a.empty() || (a.size() == 1 && !a[0]); }
```

```
    bigint operator-() const {
        bigint res = *this;
        res.sign = -sign;
        return res;
    }
    bigint abs() const {
        bigint res = *this;
        res.sign *= res.sign;
        return res;
    }
    long long longValue() const {
        long long res = 0;
        for (int i = a.size() - 1; i >= 0; i--) res = res * base + a[i];
        return res * sign;
    }
    friend bigint gcd(const bigint &a, const bigint &b) {
        return b.isZero() ? a : gcd(b, a % b);
    }
    friend bigint lcm(const bigint &a, const bigint &b) {
        return a / gcd(a, b) * b;
    }
    void read(const string &s) {
        sign = 1;
        a.clear();
        int pos = 0;
        while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
            if (s[pos] == '-') sign = -sign;
            ++pos;
        }
        for (int i = s.size() - 1; i >= pos; i -= base_digits) {
            int x = 0;
            for (int j = max(pos, i - base_digits + 1); j <= i; j++)
                x = x * 10 + s[j] - '0';
            a.push_back(x);
        }
        trim();
    }
    friend istream &operator>>(istream &stream, bigint &v) {
        string s;
        stream >> s;
        v.read(s);
        return stream;
    }
    friend ostream &operator<<(ostream &stream, const bigint &v) {
        if (v.sign == -1) stream << '-';
        stream << (v.a.empty() ? 0 : v.a.back());
        for (int i = (int)v.a.size() - 2; i >= 0; --i)
            stream << setw(base_digits) << setfill('0') << v.a[i];
        return stream;
    }
    static vector<int> convert_base(const vector<int> &a, int old_digits,
                                    int new_digits) {
        vector<long long> p(max(old_digits, new_digits) + 1);
        p[0] = 1;
```

```
    for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int)a.size(); i++) {
        cur += a[i] * p[cur_digits];
        cur_digits += old_digits;
        while (cur_digits >= new_digits) {
            res.push_back(int(cur % p[new_digits]));
            cur /= p[new_digits];
            cur_digits -= new_digits;
        }
    }
    res.push_back((int)cur);
    while (!res.empty() && !res.back()) res.pop_back();
    return res;
}
typedef vector<long long> vll;
static vll karatsubaMultiply(const vll &a, const vll &b) {
    int n = a.size();
    vll res(n + n);
    if (n <= 32) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) res[i + j] += a[i] * b[j];
        return res;
    }
    int k = n >> 1;
    vll a1(a.begin(), a.begin() + k);
    vll a2(a.begin() + k, a.end());
    vll b1(b.begin(), b.begin() + k);
    vll b2(b.begin() + k, b.end());
    vll a1b1 = karatsubaMultiply(a1, b1);
    vll a2b2 = karatsubaMultiply(a2, b2);
    for (int i = 0; i < k; i++) a2[i] += a1[i];
    for (int i = 0; i < k; i++) b2[i] += b1[i];
    vll r = karatsubaMultiply(a2, b2);
    for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];
    for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
    for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
    for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
    return res;
}
bigint operator*(const bigint &v) const {
    vector<int> a6 = convert_base(this->a, base_digits, 6);
    vector<int> b6 = convert_base(v.a, base_digits, 6);
    vll a(a6.begin(), a6.end());
    vll b(b6.begin(), b6.end());
    while (a.size() < b.size()) a.push_back(0);
    while (b.size() < a.size()) b.push_back(0);
    while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);
    vll c = karatsubaMultiply(a, b);
    bigint res;
        res.sign = sign * v.sign;
        for (int i = 0, carry = 0; i < (int)c.size(); i++) {
            long long cur = c[i] + carry;
            res.a.push_back((int)(cur % 1000000));
            carry = (int)(cur / 1000000);
        }
        res.a = convert_base(res.a, 6, base_digits);
        res.trim();
        return res;
    }
};
```

11.2 Integer Mod

```
#include "../Contest/template.cpp"
template <ll m>
struct mod_int {
    ll x;
    mod_int(ll v = 0) {
        x = v % m;
        if (x < 0) v += m;
    }
    mod_int &operator+=(mod_int const &b) {
        x += b.x;
        if (x >= m) x -= m;
        return *this;
    }
    mod_int &operator-=(mod_int const &b) {
        x -= b.x;
        if (x < 0) x += m;
        return *this;
    }
    mod_int &operator*=(mod_int const &b) {
        x = (ll)x * b.x % m;
        return *this;
    }
    friend mod_int mpow(mod_int a, ll e) {
        mod_int res = 1;
        while (e) {
            if (e & 1) res *= a;
            a *= a;
            e >>= 1;
        }
        return res;
    }
    friend mod_int inverse(mod_int a) { return mpow(a, m - 2); }
    mod_int &operator/=(mod_int const &b) { return *this *= inverse(b); }
    friend mod_int operator+(mod_int a, mod_int const b) { return a += b;
    }
    mod_int operator++(int) { return this->x = (this->x + 1) % m; }
    mod_int operator++() { return this->x = (this->x + 1) % m; }
    friend mod_int operator-(mod_int a, mod_int const b) { return a -= b;
    }
```

```cpp
    friend mod_int operator-(mod_int const a) { return 0 - a; }
    mod_int operator--(int) { return this->x = (this->x - 1 + m) % m; }
    mod_int operator--() { return this->x = (this->x - 1 + m) % m; }
    friend mod_int operator*(mod_int a, mod_int const b) { return a *= b;
    }
    friend mod_int operator/(mod_int a, mod_int const b) { return a /= b;
    }
    friend ostream &operator<<(ostream &os, mod_int const &a) {
        return os << a.x;
    }
    friend bool operator==(mod_int const &a, mod_int const &b) {
        return a.x == b.x;
    }
    friend bool operator!=(mod_int const &a, mod_int const &b) {
        return a.x != b.x;
    }
};
```

11.3 Integer Mod (complete)

```cpp
#include "../Contest/template.cpp"
template <ll Mod>
struct modint {
    static constexpr ll mod = Mod;
    ll v;
    modint() : v(0) {}
    template <ll Mod2>
    modint(const modint<Mod2> &x) : v(x.value()) {}
    modint(ll x) : v(safe_mod(x)) {}
    ll &value() { return v; }
    const ll &value() const { return v; }
    static ll safe_mod(ll x) {
        return x >= 0 ? x % mod : ((x % mod) + mod) % mod;
    }
    template <typename T>
    explicit operator T() const {
        return (T)v;
    }
    bool operator==(const modint rhs) const noexcept { return v == rhs.v;
    }
    bool operator!=(const modint rhs) const noexcept { return v != rhs.v;
    }
    bool operator<(const modint rhs) const noexcept { return v < rhs.v; }
    bool operator<=(const modint rhs) const noexcept { return v <= rhs.v;
    }
    bool operator>(const modint rhs) const noexcept { return v > rhs.v; }
    bool operator>=(const modint rhs) const noexcept { return v >= rhs.v;
    }
    modint operator++(int) {
        modint res = *this;
        *this += 1;
        return res;
    }
    modint operator--(int) {
        modint res = *this;
        *this -= 1;
        return res;
    }
    modint &operator++() { return *this += 1; }
    modint &operator--() { return *this -= 1; }
    modint operator+() const { return modint(*this); }
    modint operator-() const { return mod - modint(*this); }
    friend modint operator+(const modint lhs, const modint rhs) noexcept {
        return modint(lhs) += rhs;
    }
    friend modint operator-(const modint lhs, const modint rhs) noexcept {
        return modint(lhs) -= rhs;
    }
    friend modint operator*(const modint lhs, const modint rhs) noexcept {
        return modint(lhs) *= rhs;
    }
    friend modint operator/(const modint lhs, const modint rhs) noexcept {
        return modint(lhs) /= rhs;
    }
    modint &operator+=(const modint rhs) {
        v += rhs.v;
        if (v >= mod) v -= mod;
        return *this;
    }
    modint &operator-=(const modint rhs) {
        if (v < rhs.v) v += mod;
        v -= rhs.v;
        return *this;
    }
    modint &operator*=(const modint rhs) {
        v = v * rhs.v % mod;
        return *this;
    }
    modint &operator/=(modint rhs) { return *this *= rhs.inv(); }
    modint pow(ll p) const {
        static_assert(mod < static_cast<ll>(1) << 32,
                      "Modulus must be less than 2**32");
        modint res = 1, a = *this;
        while (p) {
            if (p & 1) res *= a;
            a *= a;
            p >>= 1;
        }
        return res;
    }
    modint inv() const { return pow(mod - 2); }
    modint sqrt() const {
        modint b = 1;
        while (b.pow((mod - 1) >> 1) == 1) b += 1;
        ll m = mod - 1, e = 0;
        while (~m & 1) m >>= 1, e++;
        auto x = pow((m - 1) >> 1);
        auto y = *this * x * x;
        x *= *this;
        auto z = b.pow(m);
        while (y != 1) {
```

```
        ll j = 0;
        for (modint t = y; t != 1; t *= t, ++j);
        z.pow(1ll << (e - j - 1));
        x *= z;
        z *= z;
        y *= z;
        e = j;
    }
    return x;
}
friend ostream &operator<<(ostream &s, const modint &x) {
    s << x.value();
    return s;
}
friend istream &operator>>(istream &s, modint &x) {
    ll value;
    s >> value;
    x = {value};
    return s;
}
};
```

11.4 Matrix

```
template <typename T>
struct Matrix {
    vector<vector<T>> d;

    Matrix() : Matrix(0) {}
    Matrix(int n) : Matrix(n, n) {}
    Matrix(int n, int m) : Matrix(vector<vector<T>>(n, vector<T>(m))) {}
    Matrix(const vector<vector<T>> &v) : d(v) {}

    constexpr int n() const { return (int)d.size(); }
    constexpr int m() const { return n() ? (int)d[0].size() : 0; }

    void rotate() { *this = rotated(); }

    Matrix<T> rotated() const {
        Matrix<T> res(m(), n());
        for (int i = 0; i < m(); i++) {
            for (int j = 0; j < n(); j++) {
                res[i][j] = d[n() - j - 1][i];
            }
        }
        return res;
    }

    Matrix<T> pow(int power) const {
        assert(n() == m());

        auto res = Matrix<T>::identity(n());
        auto b = *this;
        while (power) {
            if (power & 1) res *= b;
            b *= b;
            power >>= 1;
        }
        return res;
    }

    Matrix<T> submatrix(int start_i, int start_j, int rows = INT_MAX,
                        int cols = INT_MAX) const {
        rows = min(rows, n() - start_i);
        cols = min(cols, m() - start_j);
        if (rows <= 0 or cols <= 0) return {};

        Matrix<T> res(rows, cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                res[i][j] = d[i + start_i][j + start_j];
        return res;
    }

    Matrix<T> translated(int x, int y) const {
        Matrix<T> res(n(), m());
        for (int i = 0; i < n(); i++) {
            for (int j = 0; j < m(); j++) {
                if (i + x < 0 or i + x >= n() or j + y < 0 or j + y >= m()
)
                    continue;
                res[i + x][j + y] = d[i][j];
            }
        }
        return res;
    }

    static Matrix<T> identity(int n) {
        Matrix<T> res(n);
        for (int i = 0; i < n; i++) res[i][i] = 1;
        return res;
    }

    vector<T> &operator[](int i) { return d[i]; }
    const vector<T> &operator[](int i) const { return d[i]; }
    Matrix<T> &operator+=(T value) {
        for (auto &row : d) {
            for (auto &x : row) x += value;
        }
        return *this;
    }
    Matrix<T> operator+(T value) const {
        auto res = *this;
        for (auto &row : res) {
            for (auto &x : row) x = x + value;
        }
        return res;
    }
    Matrix<T> &operator-=(T value) {
        for (auto &row : d) {
            for (auto &x : row) x -= value;
        }
        return *this;
    }
    Matrix<T> operator-(T value) const {
        auto res = *this;
        for (auto &row : res) {
            for (auto &x : row) x = x - value;
        }
        return res;
    }
```

```
    Matrix<T> &operator*=(T value) {
        for (auto &row : d) {
            for (auto &x : row) x *= value;
        }
        return *this;
    }
    Matrix<T> operator*(T value) const {
        auto res = *this;
        for (auto &row : res) {
            for (auto &x : row) x = x * value;
        }
        return res;
    }
    Matrix<T> &operator/=(T value) {
        for (auto &row : d) {
            for (auto &x : row) x /= value;
        }
        return *this;
    }
    Matrix<T> operator/(T value) const {
        auto res = *this;
        for (auto &row : res) {
            for (auto &x : row) x = x / value;
        }
        return res;
    }
    Matrix<T> &operator+=(const Matrix<T> &o) {
        assert(n() == o.n() and m() == o.m());
        for (int i = 0; i < n(); i++) {
            for (int j = 0; j < m(); j++) {
                d[i][j] += o[i][j];
            }
        }
        return *this;
    }
    Matrix<T> operator+(const Matrix<T> &o) const {
        assert(n() == o.n() and m() == o.m());
        auto res = *this;
        for (int i = 0; i < n(); i++) {
            for (int j = 0; j < m(); j++) {
                res[i][j] = res[i][j] + o[i][j];
            }
        }
        return res;
    }
    Matrix<T> &operator-=(const Matrix<T> &o) {
        assert(n() == o.n() and m() == o.m());
        for (int i = 0; i < n(); i++) {
            for (int j = 0; j < m(); j++) {
                d[i][j] -= o[i][j];
            }
        }
        return *this;
    }
    Matrix<T> operator-(const Matrix<T> &o) const {
        assert(n() == o.n() and m() == o.m());
        auto res = *this;
        for (int i = 0; i < n(); i++) {
            for (int j = 0; j < m(); j++) {
                res[i][j] = res[i][j] - o[i][j];
            }
        }
        return res;
    }
    Matrix<T> &operator*=(const Matrix<T> &o) {
        *this = *this * o;
        return *this;
    }
    Matrix<T> operator*(const Matrix<T> &o) const {
        assert(m() == o.n());
        Matrix<T> res(n(), o.m());
        for (int i = 0; i < res.n(); i++) {
            for (int j = 0; j < res.m(); j++) {
                auto &x = res[i][j];
                for (int k = 0; k < m(); k++) {
                    x += (d[i][k] * o[k][j]);
                }
            }
        }
        return res;
    }
    friend istream &operator>>(istream &is, Matrix<T> &mat) {
        for (auto &row : mat)
            for (auto &x : row) is >> x;
        return is;
    }
    friend ostream &operator<<(ostream &os, const Matrix<T> &mat) {
        bool frow = 1;
        for (auto &row : mat) {
            if (not frow) os << '\n';
            bool first = 1;
            for (auto &x : row) {
                if (not first) os << ' ';
                os << x;
                first = 0;
            }
            frow = 0;
        }
        return os;
    }
    auto begin() { return d.begin(); }
    auto end() { return d.end(); }
    auto rbegin() { return d.rbegin(); }
    auto rend() { return d.rend(); }
    auto begin() const { return d.begin(); }
    auto end() const { return d.end(); }
    auto rbegin() const { return d.rbegin(); }
    auto rend() const { return d.rend(); }
};
```

12 Problems

12.1 2081 - Fixed-Lenght Paths II

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 2'00'000;

int N, K1, K2;
vector<int> ADJ[MAXN];

int64_t ans = 0;

int sz[MAXN], removed[MAXN];
void calcSize(int u, int p = -1) {
    sz[u] = 1;
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) {
            calcSize(v, u);
            sz[u] += sz[v];
        }
    }
}

int findCentroid(int u, int mxSz, int p = -1) {
    for (int v : ADJ[u]) {
        if (!removed[v] and v != p and sz[v] * 2 >= mxSz)
            return findCentroid(v, mxSz, u);
    }
    return u;
}

int64_t cnt[MAXN], totCnt[MAXN], initialSum;
int mxD;
void dfs(int u, int p, int d) {
    if (d > K2) return;
    cnt[d]++;
    mxD = max(mxD, d);
    if (K1 - 1 <= d and d <= K2 - 1) initialSum++;
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) {
            dfs(v, u, d + 1);
        }
    }
}

void solve(int curRoot) {
    calcSize(curRoot);
    int centroid = findCentroid(curRoot, sz[curRoot]);

    removed[centroid] = true;

    int totMxD = 0;
    initialSum = (K1 == 1);
    // cerr << "centroid: " << centroid << '\n';
    for (int v : ADJ[centroid]) {
        if (!removed[v]) {
            // cerr << "v: " << v << '\n';
            mxD = 0;
            int64_t curSum = initialSum;
            dfs(v, centroid, 1);
            totMxD = max(totMxD, mxD);
            for (int d = 1; d <= mxD; d++) {
                // cerr << "d : " << d << " curSum: " << curSum << '\n';
                ans += (curSum * cnt[d]);

                int pl = max(0, K1 - d) - 1;
                if (pl >= 0) curSum += totCnt[pl];
                int pr = K2 - d;
                curSum -= totCnt[pr];
            }
            for (int d = 1; d <= mxD; d++) totCnt[d] += cnt[d];

            fill(&cnt[1], &cnt[1] + mxD + 1, 0);
        }
    }

    // cerr << "centroid: " << centroid
    //<< " ans: " << ans << '\n';
    for (int d = 1; d <= totMxD; d++) totCnt[d] = 0;

    for (int v : ADJ[centroid])
        if (!removed[v]) solve(v);
}

int32_t main() {
    ios_base::sync_with_stdio(!cin.tie(0));
    totCnt[0] = 1;
    cin >> N >> K1 >> K2;
    for (int i = 0; i < N - 1; i++) {
        int u, v;
        cin >> u >> v;
        u--, v--;
        ADJ[u].emplace_back(v);
        ADJ[v].emplace_back(u);
    }

    solve(0);

    cout << ans << '\n';
}

// AC, centroid decomposition
```

12.2 Fixed lenght pahts I

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 2'00'000;
int N, K;
vector<int> ADJ[MAXN];

int64_t ans;
bool removed[MAXN];
int cnt[MAXN];

int sz[MAXN];

void calcSize(int u, int p = -1) {
    sz[u] = 1;
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) {
            calcSize(v, u);
            sz[u] += sz[v];
```

```
        }
    }
}
int getCentroid(int mxSz, int u, int p = -1) {
    for (int v : ADJ[u]) {
        if (v != p and !removed[v] and sz[v] >= mxSz)
            return getCentroid(mxSz, v, u);
    }
    return u;
}
int mxd;
void dfs(int u, int p, bool upd, int d = 1) {
    if (d > K) return;
    mxd = max(mxd, d);
    upd ? cnt[d]++ : ans += cnt[K - d];
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) dfs(v, u, upd, d + 1);
    }
}
void solve(int u) {
    calcSize(u);
    int c = getCentroid(sz[u] >> 1, u);
    removed[c] = true;
    mxd = 0;
    cnt[0] = 1;
    for (int v : ADJ[c]) {
        if (!removed[v]) {
            dfs(v, c, false);
            dfs(v, c, true);
        }
    }
    for (int i = 0; i <= mxd; i++) cnt[i] = 0;
    for (int v : ADJ[c]) {
        if (!removed[v]) solve(v);
    }
}
int32_t main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cin >> N >> K;
    for (int i = 0; i < N - 1; i++) {
        int u, v;
        cin >> u >> v;
        u--, v--;
        ADJ[u].emplace_back(v);
        ADJ[v].emplace_back(u);
    }
    solve(0);
    cout << ans << '\n';
    return 0;
}
```

12.3 Fixed lenght paths II

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 2'00'000;
int N, K1, K2;
vector<int> ADJ[MAXN];
int64_t ans = 0;
int sz[MAXN], removed[MAXN];
void calcSize(int u, int p = -1) {
    sz[u] = 1;
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) {
            calcSize(v, u);
            sz[u] += sz[v];
        }
    }
}
int findCentroid(int u, int mxSz, int p = -1) {
    for (int v : ADJ[u]) {
        if (!removed[v] and v != p and sz[v] * 2 >= mxSz)
            return findCentroid(v, mxSz, u);
    }
    return u;
}
int64_t cnt[MAXN], totCnt[MAXN], initialSum;
int mxD;
void dfs(int u, int p, int d) {
    if (d > K2) return;
    cnt[d]++;
    mxD = max(mxD, d);
    if (K1 - 1 <= d and d <= K2 - 1) initialSum++;
    for (int v : ADJ[u]) {
        if (v != p and !removed[v]) {
            dfs(v, u, d + 1);
        }
    }
}
void solve(int curRoot) {
    calcSize(curRoot);
    int centroid = findCentroid(curRoot, sz[curRoot]);
    removed[centroid] = true;
    int totMxD = 0;
    initialSum = (K1 == 1);
    // cerr << "centroid: " << centroid << '\n';
    for (int v : ADJ[centroid]) {
        if (!removed[v]) {
            // cerr << "v: " << v << '\n';
            mxD = 0;
            int64_t curSum = initialSum;
            dfs(v, centroid, 1);
            totMxD = max(totMxD, mxD);
            for (int d = 1; d <= mxD; d++) {
                // cerr << "d : " << d << " curSum: " << curSum << '\n';
```

```
            ans += (curSum * cnt[d]);
            int pl = max(0, K1 - d) - 1;
            if (pl >= 0) curSum += totCnt[pl];
            int pr = K2 - d;
            curSum -= totCnt[pr];
        }
        for (int d = 1; d <= mxD; d++) totCnt[d] += cnt[d];
        fill(&cnt[1], &cnt[1] + mxD + 1, 0);
    }
    }
    // cerr << "centroid: " << centroid
    //<< " ans: " << ans << '\n';
    for (int d = 1; d <= totMxD; d++) totCnt[d] = 0;
    for (int v : ADJ[centroid])
        if (!removed[v]) solve(v);
}
int32_t main() {
    ios_base::sync_with_stdio(!cin.tie(0));
    totCnt[0] = 1;
    cin >> N >> K1 >> K2;
    for (int i = 0; i < N - 1; i++) {
        int u, v;
        cin >> u >> v;
        u--, v--;
        ADJ[u].emplace_back(v);
        ADJ[v].emplace_back(u);
    }
    solve(0);
    cout << ans << '\n';
}
// AC, centroid decomposition
```

13 Strings

13.1 Z-Function

13.1.1 Z-function building

Description: The Z-function is an algorithm used to compute the Z-array of a given string.For a string s, $Z[i]$ represents the length of the longest common prefix between the string s and the suffix of s starting from the index i.
Usage: The function z_function_build(s) takes a single argument s, which is a string (or any container-like structure), and returns a vector of integers representing the Z-function of the input.

```
string s = "abacaba";
vector<int> result = z_function_build(s);
// result = [0, 0, 1, 0, 3, 0, 1]
vector<int> v = {1, 2, 3, 1, 2, 3};
vector<int> result_v = z_function_build(v);
// result_v = [0, 1, 2, 3, 0, 1]
```

Time: $O(n)$
Memory: $O(n)$
Warning: By definition $Z[0] = 0$, remember to treat it appart.

```
#pragma once
#include "../../Contest/template.cpp"
template <typename Seq>
vector<int> z_function_build(const Seq& s) {
    int n = int(s.size());
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}
```

13.2 Count distinct anagrams

```
const ll MOD = 1e9 + 7;
const int maxn = 1e6;
vll fs(maxn + 1);
void precompute() {
    fs[0] = 1;
    for (ll i = 1; i <= maxn; i++) {
        fs[i] = (fs[i - 1] * i) % MOD;
    }
}
ll fpow(ll a, int n, ll mod = LLONG_MAX) {
    if (n == 0) return 1;
    if (n == 1) return a;
    ll x = fpow(a, n / 2, mod) % mod;
    return ((x * x) % mod * (n & 1 ? a : 1ll)) % mod;
}
ll distinctAnagrams(const string &s) {
    precompute();
    vi hist('z' - 'a' + 1, 0);
    for (auto &c : s) hist[c - 'a']++;
    ll ans = fs[len(s)];
    for (auto &q : hist) {
        ans = (ans * fpow(fs[q], MOD - 2, MOD)) % MOD;
    }
    return ans;
}
```

13.3 Double hash range query

```
#include "../Contest/template.cpp"
using ll = long long;
using vll = vector<ll>;
using pll = pair<ll, ll>;
```

```cpp
const int MAXN(1'000'000);
const ll MOD = 1000027957;
const ll MOD2 = 1000015187;
const ll P = 31;
ll p[MAXN + 1], p2[MAXN + 1];
void precompute() {
    p[0] = p2[0] = 1;
    for (int i = 1; i <= MAXN; i++)
        p[i] = (P * p[i - 1]) % MOD, p2[i] = (P * p2[i - 1]) % MOD2;
}
struct Hash {
    int n;
    vll h, h2, hi, hi2;
    Hash() {}
    Hash(const string &s) : n(s.size()), h(n), h2(n), hi(n), hi2(n) {
        h[0] = h2[0] = s[0];
        for (int i = 1; i < n; i++)
            h[i] = (s[i] + h[i - 1] * P) % MOD,
            h2[i] = (s[i] + h2[i - 1] * P) % MOD2;
        hi[n - 1] = hi2[n - 1] = s[n - 1];
        for (int i = n - 2; i >= 0; i--)
            hi[i] = (s[i] + hi[i + 1] * P) % MOD,
            hi2[i] = (s[i] + hi2[i + 1] * P) % MOD2;
    }
    pll query(int l, int r) {
        ll hash = (h[r] - (l ? h[l - 1] * p[r - l + 1] % MOD : 0));
        ll hash2 = (h2[r] - (l ? h2[l - 1] * p2[r - l + 1] % MOD2 : 0));
        return {(hash < 0 ? hash + MOD : hash),
                (hash2 < 0 ? hash2 + MOD2 : hash2)};
    }
    pll query_inv(int l, int r) {
        ll hash = (hi[l] - (r + 1 < n ? hi[r + 1] * p[r - l + 1] % MOD :
    0));
        ll hash2 =
            (hi2[l] - (r + 1 < n ? hi2[r + 1] * p2[r - l + 1] % MOD2 : 0))
    ;
        return {(hash < 0 ? hash + MOD : hash),
                (hash2 < 0 ? hash2 + MOD2 : hash2)};
    }
};
```

13.4 Hash range query

```cpp
#include "../Contest/template.cpp"
const ll P = 31;
const ll MOD = 1e9 + 9;
const int MAXN(1e6);
ll ppow[MAXN + 1];
void pre_calc() {
    ppow[0] = 1;
    for (int i = 1; i <= MAXN; i++) ppow[i] = (ppow[i - 1] * P) % MOD;
}
struct Hash {
    int n;
    vll h, hi;
    Hash(const string &s) : n(s.size()), h(n), hi(n) {
        h[0] = s[0];
        hi[n - 1] = s[n - 1];
        for (int i = 1; i < n; i++) {
            h[i] = (s[i] + h[i - 1] * P) % MOD;
            hi[n - i - 1] = (s[n - i - 1] + hi[n - i - 1] * P) % MOD;
        }
    }
    ll qry(int l, int r) {
        ll hash = (h[r] - (l ? h[l - 1] * ppow[r - l + 1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    ll qry_inv(int l, int r) {
        ll hash = (hi[l] - (r + 1 < n ? hi[r + 1] * ppow[r - l + 1] % MOD
: 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

13.5 Hash unsigned long long $2^{64}-1$

Description: Arithmetic mod $2^{64}-1$. 2x slower than mod 2^{64} and more code, but works on evil test data (e.g. Thue-Morse, where ABBA... and BAAB... of length 2^{10} hash the same mod 2^{64}).
"typedef ull H;" instead if you think test data is random.

```cpp
#include "../Contest/template.cpp"
struct H {
    ull x;
    H(ull x = 0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) {
        auto m = (__uint128_t)x * o.x;
        return H((ull)m) + (ull)(m >> 64);
    }
    ull get() const { return x + !~x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (long long)1e11 + 3;  // (order ~ 3e9; random also ok)
struct Hash {
    int n;
    vector<H> ha, pw;
    Hash(string &str) : n(str.size()), ha((int)str.size() + 1), pw(ha) {
        pw[0] = 1;
        for (int i = 0; i < (int)str.size(); i++)
            ha[i + 1] = ha[i] * C + str[i], pw[i + 1] = pw[i] * C;
    }
    H query(int a, int b) {  // hash [a, b]
        b++;
        return ha[b] - ha[a] * pw[b - a];
```

```
    }
};
vector<H> getHashes(string &str, int length) {
    if ((int)str.size() < length) return {};
    H h = 0, pw = 1;
    for (int i = 0; i < length; i++) h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    for (int i = length; i < (int)str.size(); i++)
        ret.push_back(h = h * C + str[i] - pw * str[i - length]);
    return ret;
}
H hashString(string &s) {
    H h{};
    for (char c : s) h = h * C + c;
    return h;
}
```

13.6 K-th digit in digit string

Description: Find the k-th digit in a *digit string*, only works for $1 <= k <= 10^{18}$!
Time: precompute $O(1)$, query $O(1)$

```
using vull = vector<ull>;
vull pow10;
vector<array<ull, 4>> memo;
void precompute(int maxpow = 18) {
    ull qtd = 1;
    ull start = 1;
    ull end = 9;
    ull curlenght = 9;
    ull startstr = 1;
    ull endstr = 9;
    for (ull i = 0, j = 1ll; (int)i < maxpow; i++, j *= 10ll) pow10.eb(j);
    for (ull i = 0; i < maxpow - 1ull; i++) {
        memo.push_back({start, end, startstr, endstr});
        start = end + 1ll;
        end = end + (9ll * pow10[qtd]);
        curlenght = end - start + 1ull;
        qtd++;
        startstr = endstr + 1ull;
        endstr = (endstr + 1ull) + (curlenght)*qtd - 1ull;
    }
}
char kthDigit(ull k) {
    int qtd = 1;
    for (auto [s, e, ss, es] : memo) {
        if (k >= ss and k <= es) {
            ull pos = k - ss;
            ull index = pos / qtd;
            ull nmr = s + index;
            int i = k - ss - qtd * index;
            return ((nmr / pow10[qtd - i - 1]) % 10) + '0';
        }
        qtd++;
    }
    return 'X';
}
```

13.7 KMP

```
/**
 * Author: Johan Sannemo
 * Date: 2016-12-15
 * License: CC0
 * Description: pi[x] computes the length of the longest prefix of s that
   ends
 * at x, other than s[0...x] itself (abacaba -> 0010123). Can be used to
   find
 * all occurrences of a string. Time: O(n) Status: Tested on
 * kattis:stringmatching
 */
/*
 * @Title: Prefix function - Knuth-Morris-Pratt
 * @Description: Given a string $S$ builds an array $A$ such that
 * $A_i$ is the longest suffix that ends in $i$ and is also a prefix
 * of $S$.
 *
 *
 */
#pragma once
vi pi(const string& s) {
    vi p(sz(s));
    rep(i, 1, sz(s)) {
        int g = p[i - 1];
        while (g && s[i] != s[g]) g = p[g - 1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}
vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p) - sz(s), sz(p)) if (p[i] == sz(pat))
        res.push_back(i - 2 * sz(pat));
    return res;
}
```

13.8 Longest Palindrome Substring (Manacher)

Description: Finds the longest palindrome substring, manacher returns a vector where the i-th position is how much is possible to grow the string to the left and the right of i and keep it a palindrome.
Time: $O(N)$

```
vi manacher(const string &s) {
    int n = len(s) - 2;
    vi p(n + 2);
    int l = 1, r = 1;
```

```
    for (int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while (s[i - p[i]] == s[i + p[i]]) p[i]++;
        if (i + p[i] > r) l = i - p[i], r = i + p[i];
        p[i]--;
    }
    return p;
}
string longest_palindrome(const string &s) {
    string t("$#");
    for (auto c : s) t.push_back(c), t.push_back('#');
    t.push_back('^');
    vi xs = manacher(t);
    int mpos = max_element(all(xs)) - xs.begin();
    string p;
    for (int k = xs[mpos], i = mpos - k; i <= mpos + k; i++)
        if (t[i] != '#') p.push_back(t[i]);
    return p;
}
```

13.9 Longest palindrome

```
string longest_palindrome(const string &s) {
    int n = (int)s.size();
    vector<array<int, 2>> dp(n);
    pii odd(0, -1), even(0, -1);
    pii ans;
    for (int i = 0; i < n; i++) {
        int k = 0;
        if (i > odd.second)
            k = 1;
        else
            k = min(dp[odd.first + odd.second - i][0], odd.second - i + 1)
    ;
        while (i - k >= 0 and i + k < n and s[i - k] == s[i + k]) k++;
        dp[i][0] = k--;
        if (i + k > odd.second) odd = {i - k, i + k};
        if (2 * dp[i][0] - 1 > ans.second) ans = {i - k, 2 * dp[i][0] -
    1};
        k = 0;
        if (i <= even.second)
            k = min(dp[even.first + even.second - i + 1][1],
                    even.second - i + 1);
        while (i - k - 1 >= 0 and i + k < n and s[i - k - 1] == s[i + k])
    k++;
        dp[i][1] = k--;
        if (i + k > even.second) even = {i - k - 1, i + k};
        if (2 * dp[i][1] > ans.second) ans = {i - k - 1, 2 * dp[i][1]};
    }
    return s.substr(ans.first, ans.second);
}
```

13.10 Lyndon factorization

```
vi lyndon_factorization(string S) {
    auto sa = suffix_array(S);
    vi ans;
    vi mex(len(S) + 1, 0);
    int p = 0;
    rtrav(si, sa) {
        if (si == p) {
            ans.eb(si);
        }
        mex[si] = 1;
        while (mex[p]) p++;
    }
    ans.eb(len(S));
    return ans;
}
```

13.11 Rabin-Karp

```
size_t rabin_karp(const string &s, const string &p) {
    if (s.size() < p.size()) return 0;
    auto n = s.size(), m = p.size();
    const ll p1 = 31, p2 = 29, q1 = 1e9 + 7, q2 = 1e9 + 9;
    const ll p1_1 = fpow(p1, q1 - 2, q1), p1_2 = fpow(p1, m - 1, q1);
    const ll p2_1 = fpow(p2, q2 - 2, q2), p2_2 = fpow(p2, m - 1, q2);
    pair<ll, ll> hs, hp;
    for (int i = (int)m - 1; ~i; --i) {
        hs.first = (hs.first * p1) % q1;
        hs.first = (hs.first + (s[i] - 'a' + 1)) % q1;
        hs.second = (hs.second * p2) % q2;
        hs.second = (hs.second + (s[i] - 'a' + 1)) % q2;
        hp.first = (hp.first * p1) % q1;
        hp.first = (hp.first + (p[i] - 'a' + 1)) % q1;
        hp.second = (hp.second * p2) % q2;
        hp.second = (hp.second + (p[i] - 'a' + 1)) % q2;
    }
    size_t occ = 0;
    for (size_t i = 0; i < n - m; i++) {
        occ += (hs == hp);
        int fi = s[i] - 'a' + 1;
        int fm = s[i + m] - 'a' + 1;
        hs.first = (hs.first - fi + q1) % q1;
        hs.first = (hs.first * p1_1) % q1;
        hs.first = (hs.first + fm * p1_2) % q1;
        hs.second = (hs.second - fi + q2) % q2;
        hs.second = (hs.second * p2_1) % q2;
        hs.second = (hs.second + fm * p2_2) % q2;
    }
    occ += hs == hp;
    return occ;
}
```

13.12 Suffix Automaton

Description: A suffix automaton A for a string s is a minimal finite automaton that recognizes the suffixes of s.

```
#pragma once
#include "../Contest/template.cpp"
struct SuffixAutomaton {
    int n;
    vi suffix_link, max_len;
    vi2d transition;
    SuffixAutomaton(const string &s, int alphabet_size='z'-'a'+1, int norm
     = 'a') : n(len(s), suffix_link(n<<1), max_len(n<<1) , transition(n
    <<1, vi(alphabet_size, -1){
        int last = 0;
        trav(c, s) {
            int max_len_cur = max_len[last] + 1;
            while
        }
    }
};
```

13.13 Suffix array

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif
#define endl '\n'
#define fastio                          \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
#define int long long
#define all(j) j.begin(), j.end()
#define rall(j) j.rbegin(), j.rend()
#define len(j) (int)j.size()
#define rep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i < (b); i++)
#define rrep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i > (b); i--)
#define trav(xi, xs) for (auto &xi : xs)
#define rtrav(xi, xs) for (auto &xi : ranges::views::reverse(xs))
#define pb push_back
#define pf push_front
#define ppb pop_back
#define ppf pop_front
#define eb emplace_back
#define lb lower_bound
#define ub upper_bound
#define fi first
#define se second
#define emp emplace
#define ins insert
#define divc(a, b) ((a) + (b) - 1ll) / (b)
using str = string;
using ll = long long;
using ull = unsigned long long;
using ld = long double;
using vll = vector<ll>;
using pll = pair<ll, ll>;
using vll2d = vector<vll>;
using vi = vector<int>;
using vi2d = vector<vi>;
using pii = pair<int, int>;
using vpii = vector<pii>;
using vc = vector<char>;
using vs = vector<str>;
template <typename T, typename T2>
using umap = unordered_map<T, T2>;
template <typename T>
using pqmn = priority_queue<T, vector<T>, greater<T>>;
template <typename T>
using pqmx = priority_queue<T, vector<T>>;
template <typename T, typename U>
inline bool chmax(T &a, U const &b) {
    return (a < b ? a = b, 1 : 0);
}
template <typename T, typename U>
inline bool chmin(T &a, U const &b) {
    return (a > b ? a = b, 1 : 0);
}
vector<int> sort_cyclic_shifts(string const &s) {
    int n = s.size();
    const int alphabet = 128;
    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]]) classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0) pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++) cnt[i] += cnt[i - 1];
        for (int i = n - 1; i >= 0; i--) p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
```

```cpp
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
            pair<int, int> prev = {c[p[i - 1]], c[(p[i - 1] + (1 << h)) %
    n]};
            if (cur != prev) ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}
vector<int> suffix_array(string s) {
    s += "$";
    vector<int> p = sort_cyclic_shifts(s);
    p.erase(p.begin());
    return p;
}
vector<int> longestCommonPrefix(const string &s, const vector<int> &suf) {
    int n = s.size();
    vector<int> isuf(n), res(n - 1);
    for (int i = 0; i < n; ++i) isuf[suf[i]] = i;
    int k = 0;
    for (; isuf[k] != n - 1; ++k) {
        int cmp_i = suf[isuf[k] + 1];
        int r = k == 0 ? 0 : max(res[isuf[k - 1]] - 1, (int)0);
        while (k + r < n && cmp_i + r < n && s[k + r] == s[cmp_i + r]) ++r
    ;
        res[isuf[k]] = r;
    }
    ++k;
    for (int i = k; i < n; ++i) {
        int cmp_i = suf[isuf[i] + 1];
        int r = i == k ? 0 : max(res[isuf[i - 1]] - 1, (int)0);
        while (i + r < n && cmp_i + r < n && s[i + r] == s[cmp_i + r]) ++r
    ;
        res[isuf[i]] = r;
    }
    return res;
}
ll distinct_substrings(const string &s, const vi &sa) {
    int n = len(s);
    vi lcp = longestCommonPrefix(s, sa);
    ll ans = n - sa[0];
    rep(i, 1, n) { ans += n - sa[i] - lcp[i - 1]; }
    return ans;
}
void run();
int32_t main() {
#ifndef LOCAL
    fastio;
#endif
    int T = 1;
    /*cin >> T;*/
    rep(t, 0, T) {
        dbg(t);
        run();
    }
}
void run() {
    string S;
    cin >> S;
    auto sa = suffix_array(S);
    cout << distinct_substrings(S, sa) << endl;
}
```

13.14 Suffix array (supreme)

```cpp
template <typename T = ll,
          auto cmp = [](T &src1, T &src2, T &dst) { dst = min(src1, src2);
     }>
class SparseTable {
   private:
    int sz;
    vi logs;
    vector<vector<T>> st;
   public:
    SparseTable() {}
    SparseTable(const vector<T> &v) : sz(len(v)), logs(sz + 1) {
        rep(i, 2, sz + 1) logs[i] = logs[i >> 1] + 1;
        st.resize(logs[sz] + 1, vector<T>(sz));
        rep(i, 0, sz) st[0][i] = v[i];
        for (int k = 1; (1 << k) <= sz; k++) {
            for (int i = 0; i + (1 << k) <= sz; i++) {
                cmp(st[k - 1][i], st[k - 1][i + (1 << (k - 1))], st[k][i])
    ;
            }
        }
    }
    T query(int l, int r) {
        r++;
        const int k = logs[r - l];
        T ret;
        cmp(st[k][l], st[k][r - (1 << k)], ret);
        return ret;
    }
};
template <typename T>
using RMQ = SparseTable<T, [](T &a, T &b, T &c) { c = min(a, b); }>;
// éCrditos: ShahjalalShohag
// O(N)
struct SA {
    string s;
    int n;
    vector<int> sa, lcp, pos;
    RMQ<int> rmq;
    void induced_sort(vector<int> &vec, int val, vector<int> &sa,
                      vector<bool> &sl, vector<int> &lms) {
        vector<int> l(val), r(val);
        for (int c : vec) {
```

```
        if (c + 1 < val) l[c + 1]++;
        r[c]++;
    }
    partial_sum(l.begin(), l.end(), l.begin());
    partial_sum(r.begin(), r.end(), r.begin());
    fill(sa.begin(), sa.end(), -1);
    for (int i = lms.size() - 1; i >= 0; i--) sa[--r[vec[lms[i]]]] =
lms[i];
    for (int i : sa) {
        if (i >= 1 && sl[i - 1]) sa[l[vec[i - 1]]++] = i - 1;
    }
    fill(r.begin(), r.end(), 0);
    for (int c : vec) r[c]++;
    partial_sum(r.begin(), r.end(), r.begin());
    for (int k = sa.size() - 1, i = sa[k]; k >= 1; --k, i = sa[k]) {
        if (i >= 1 && !sl[i - 1]) sa[--r[vec[i - 1]]] = i - 1;
    }
}
vector<int> build_sa(vector<int> &vec, int val) {
    int n = vec.size();
    vector<int> sa(n), lms;
    vector<bool> sl(n);
    sl[n - 1] = false;
    for (int i = n - 2; i >= 0; i--) {
        sl[i] =
            (vec[i] > vec[i + 1] || (vec[i] == vec[i + 1] && sl[i +
1]));
        if (sl[i] && !sl[i + 1]) lms.push_back(i + 1);
    }
    reverse(lms.begin(), lms.end());
    induced_sort(vec, val, sa, sl, lms);
    vector<int> new_lms(lms.size()), lms_vec(lms.size());
    for (int i = 0, k = 0; i < n; i++) {
        if (!sl[sa[i]] && sa[i] >= 1 && sl[sa[i] - 1]) new_lms[k++] =
sa[i];
    }
    int cur = 0;
    sa[n - 1] = cur;
    for (int k = 1; k < (int)new_lms.size(); k++) {
        int i = new_lms[k - 1], j = new_lms[k];
        if (vec[i] != vec[j]) {
            sa[j] = ++cur;
            continue;
        }
        bool flag = false;
        for (int a = i + 1, b = j + 1;; ++a, ++b) {
            if (vec[a] != vec[b]) {
                flag = true;
                break;
            }
            if ((!sl[a] && sl[a - 1]) || (!sl[b] && sl[b - 1])) {
                flag = !((!sl[a] && sl[a - 1]) && (!sl[b] && sl[b -
1]));
                break;
            }
        }
```

```
            sa[j] = (flag ? ++cur : cur);
        }
        for (int i = 0; i < (int)lms.size(); i++) lms_vec[i] = sa[lms[i]];
        if (cur + 1 < (int)lms.size()) {
            auto lms_sa = build_sa(lms_vec, cur + 1);
            for (int i = 0; i < (int)lms.size(); i++)
                new_lms[i] = lms[lms_sa[i]];
        }
        induced_sort(vec, val, sa, sl, new_lms);
        return sa;
    }
    vector<int> suffix_array() {
        vector<int> vec(n + 1);
        copy(begin(s), end(s), begin(vec));
        vec.back() = '$';
        auto sa = build_sa(vec, 256);
        sa.erase(sa.begin());
        return sa;
    }
    vector<int> build_lcp() {
        int n = (int)s.size(), k = 0;
        vector<int> rank(n), lcp(n);
        for (int i = 0; i < n; i++) rank[sa[i]] = i;
        for (int i = 0; i < n; i++, k -= !!k) {
            if (rank[i] == n - 1) {
                k = 0;
                continue;
            }
            int j = sa[rank[i] + 1];
            while (i + k < n && j + k < n && s[i + k] == s[j + k]) k++;
            lcp[rank[i]] = k;
        }
        return lcp;
    }
    SA() {}
    SA(string _s) : s(_s), n(len(s)), pos(n) {
        sa = suffix_array();
        lcp = build_lcp();
        rmq = RMQ<int>(lcp);
        for (int i = 0; i < n; i++) pos[sa[i]] = i;
    }
    int get_lcp(int i,
                int j) {  // lcp na çãposio i, indica o lcp
                          // das çõposies i e i+1 do sa
        if (i == j) return n - i;
        int l = pos[i], r = pos[j];
        if (l > r) swap(l, r);
        return rmq.query(l, r);
    }
    // string s = a + '+' + b;
    tuple<int, int, int> lcs(int n) {  // m é o tamanho da string a
        int m = len(s) - n - 1;
        int best_len = 0;
```

```
        int index_s = 0;
        int index_t = 0;
        for (int i = 0; i < n + m; ++i) {
            if ((sa[i] < n && sa[i + 1] >= n + 1) ||
                (sa[i] >= n + 1 && sa[i + 1] < n)) {
                if (lcp[i] > best_len) {
                    best_len = lcp[i];
                    index_s = min(sa[i], sa[i + 1]);
                    index_t = max(sa[i], sa[i + 1]) - n - 1;
                }
            }
        }
        /*int maior = 0, pos = -1;*/
        /*for (int i = 2; i < n; i++) {*/
        /*  if ((sa[i] < n) != (sa[i - 1] < n)) {*/
        /*    if (lcp[i - 1] > maior)*/
        /*      maior = lcp[i - 1], pos = sa[i];*/
        /*  }*/
        /*}*/
        /*return {maior, pos};*/
        return {best_len, index_s, index_t};
    }
    ll distinct_subs() {  // n*(n+1)/2 - sum(lcp[i])
        ll resp = (ll)n * ((ll)n + 1) / 2;
        return resp - accumulate(lcp.begin(), lcp.end(), 0LL);
    }
};
```

13.15 Suffix automaton

```
#include <bits/stdc++.h>
using namespace std;
#ifdef LOCAL
#include "debug.cpp"
#else
#define dbg(...)
#endif
#define endl '\n'
#define fastio                      \
    ios_base::sync_with_stdio(0); \
    cin.tie(0);
#define int long long
#define all(j) j.begin(), j.end()
#define rall(j) j.rbegin(), j.rend()
#define len(j) (int)j.size()
#define rep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i < (b); i++)
#define rrep(i, a, b) \
    for (common_type_t<decltype(a), decltype(b)> i = (a); i > (b); i--)
#define trav(xi, xs) for (auto &xi : xs)
#define rtrav(xi, xs) for (auto &xi : ranges::views::reverse(xs))
#define pb push_back
#define pf push_front
#define ppb pop_back
#define ppf pop_front
#define eb emplace_back
#define lb lower_bound
#define ub upper_bound
#define fi first
#define se second
#define emp emplace
#define ins insert
#define divc(a, b) ((a) + (b) - 1ll) / (b)
using str = string;
using ll = long long;
using ull = unsigned long long;
using ld = long double;
using vll = vector<ll>;
using pll = pair<ll, ll>;
using vll2d = vector<vll>;
using vi = vector<int>;
using vi2d = vector<vi>;
using pii = pair<int, int>;
using vpii = vector<pii>;
using vc = vector<char>;
using vs = vector<str>;
template <typename T, typename T2>
using umap = unordered_map<T, T2>;
template <typename T>
using pqmn = priority_queue<T, vector<T>, greater<T>>;
template <typename T>
using pqmx = priority_queue<T, vector<T>>;
template <typename T, typename U>
inline bool chmax(T &a, U const &b) {
    return (a < b ? a = b, 1 : 0);
}
template <typename T, typename U>
inline bool chmin(T &a, U const &b) {
    return (a > b ? a = b, 1 : 0);
}
struct SuffixAutomaton {
    struct state {
        int len, link, cnt, firstpos;
        // this can be optimized using a vector with
        // the alphabet size
        map<char, int> next;
        vi inv_link;
    };
    vector<state> st;
    int sz = 0;
    int last;
    vc cloned;
    SuffixAutomaton(const string &s, int maxlen)
        : st(maxlen * 2), cloned(maxlen * 2) {
        st[0].len = 0;
        st[0].link = -1;
        sz++;
        last = 0;
        for (auto &c : s) add_char(c);
```

```cpp
    // precompute for count occurences
    for (int i = 1; i < sz; i++) {
        st[i].cnt = !cloned[i];
    }
    vector<pair<state, int>> aux;
    for (int i = 0; i < sz; i++) {
        aux.push_back({st[i], i});
    }
    sort(all(aux),
         [](const pair<state, int> &a, const pair<state, int> &b) {
             return a.fi.len > b.fi.len;
         });
    for (auto &[stt, id] : aux) {
        if (stt.link != -1) {
            st[stt.link].cnt += st[id].cnt;
        }
    }
    // for find every occurende position
    for (int v = 1; v < sz; v++) {
        st[st[v].link].inv_link.push_back(v);
    }
}
void add_char(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    st[cur].firstpos = st[cur].len - 1;
    int p = last;
    // follow the suffix link until find a
    // transition to c
    while (p != -1 and !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    // there was no transition to c so create and
    // leave
    if (p == -1) {
        st[cur].link = 0;
        last = cur;
        return;
    }
    int q = st[p].next[c];
    if (st[p].len + 1 == st[q].len) {
        st[cur].link = q;
    } else {
        int clone = sz++;
        cloned[clone] = true;
        st[clone].len = st[p].len + 1;
        st[clone].next = st[q].next;
        st[clone].link = st[q].link;
        st[clone].firstpos = st[q].firstpos;
        while (p != -1 and st[p].next[c] == q) {
            st[p].next[c] = clone;
            p = st[p].link;
        }
        st[q].link = st[cur].link = clone;
        }
        last = cur;
    }
    bool checkOccurrence(const string &t) {  // O(len(t))
        int cur = 0;
        for (auto &c : t) {
            if (!st[cur].next.count(c)) return false;
            cur = st[cur].next[c];
        }
        return true;
    }
    ll totalSubstrings() {  // distinct, O(len(s))
        ll tot = 0;
        for (int i = 1; i < sz; i++) {
            tot += st[i].len - st[st[i].link].len;
        }
        return tot;
    }
    // count occurences of a given string t
    int countOccurences(const string &t) {
        int cur = 0;
        for (auto &c : t) {
            if (!st[cur].next.count(c)) return 0;
            cur = st[cur].next[c];
        }
        return st[cur].cnt;
    }
    // find the first index where t appears a
    // substring O(len(t))
    int firstOccurence(const string &t) {
        int cur = 0;
        for (auto c : t) {
            if (!st[cur].next.count(c)) return -1;
            cur = st[cur].next[c];
        }
        return st[cur].firstpos - len(t) + 1;
    }
    vi everyOccurence(const string &t) {
        int cur = 0;
        for (auto c : t) {
            if (!st[cur].next.count(c)) return {};
            cur = st[cur].next[c];
        }
        vi ans;
        getEveryOccurence(cur, len(t), ans);
        return ans;
    }
    void getEveryOccurence(int v, int P_length, vi &ans) {
        if (!cloned[v]) ans.pb(st[v].firstpos - P_length + 1);
        for (int u : st[v].inv_link) getEveryOccurence(u, P_length, ans);
    }
};
void run();
int32_t main() {
#ifndef LOCAL
```

```
    fastio;
#endif
    int T = 1;
    /*cin >> T;*/
    rep(t, 0, T) {
        dbg(t);
        run();
    }
}
void run() {
    string S;
    cin >> S;
    SuffixAutomaton sa(S, len(S));
    cout << sa.totalSubstrings() << endl;
}
```

13.16 Suffix-Tree (Ukkonen's Algorithm)

```
/**
 * Author: Unknown
 * Date: 2017-05-15
 * Source: https://e-maxx.ru/algo/ukkonen
 * Description: Ukkonen's algorithm for online suffix tree construction.
 *  Each node contains indices [l, r) into the string, and a list of child
 * nodes. Suffixes are given by traversals of this tree, joining [l, r)
 * substrings. The root is 0 (has l = -1, r = 0), non-existent children
   are -1.
 *  To get a complete tree, append a dummy symbol -- otherwise it may
   contain
 *  an incomplete path (still useful for substring matching, though).
 * Time: $O(26N)$
 * Status: stress-tested a bit
 */
#pragma once
#include "../Contest/template.cpp"
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 };  // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a;  // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v = 0, q = 0, m = 2;
    void ukkadd(int i, int c) {
    suff:
        if (r[v] <= q) {
            if (t[v][c] == -1) {
                t[v][c] = m;
                l[m] = i;
                p[m++] = v;
                v = s[v];
                q = r[v];
                goto suff;
            }
            v = t[v][c];
            q = l[v];
        }
        if (q == -1 || c == toi(a[q]))
            q++;
        else {
            l[m + 1] = i;
            p[m + 1] = m;
            l[m] = l[v];
            r[m] = q;
            p[m] = p[v];
            t[m][c] = m + 1;
            t[m][toi(a[q])] = v;
            l[v] = q;
            p[v] = m;
            t[p[m]][toi(a[l[m]])] = m;
            v = s[p[m]];
            q = l[m];
            while (q < r[m]) {
                v = t[v][toi(a[q])];
                q += r[v] - l[v];
            }
            if (q == r[m])
                s[m] = v;
            else
                s[m] = m + 2;
            q = r[v] - (q - r[m]);
            m += 2;
            goto suff;
        }
    }
    SuffixTree(string a) : a(a) {
        fill(r, r + N, len(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1], t[1] + ALPHA, 0);
        s[0] = 1;
        l[0] = l[1] = -1;
        r[0] = r[1] = p[0] = p[1] = 0;
        rep(i, 0, len(a)) ukkadd(i, toi(a[i]));
    }
    // example: find longest common substring (uses ALPHA = 28)
    pii best;
    int lcs(int node, int i1, int i2, int olen) {
        if (l[node] <= i1 && i1 < r[node]) return 1;
        if (l[node] <= i2 && i2 < r[node]) return 2;
        int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
        rep(c, 0, ALPHA) if (t[node][c] != -1) mask |=
            lcs(t[node][c], i1, i2, len);
        if (mask == 3) best = max(best, {len, r[node] - len});
        return mask;
    }
    static pii LCS(string s, string t) {
        SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
        st.lcs(0, len(s), len(s) + 1 + len(t), 0);
        return st.best;
    }
};
```

13.17 Trie

Description:

- build with the size of the alphabet ($sigma$) and the first char ($norm$)
- $insert(s)$ insert the string in the trie $O(|s| * sigma)$
- $erase(s)$ remove the string from the trie $O(|s|)$
- $find(s)$ return the last node from the string s, 0 if not found $O(|s|)$

```cpp
#include "../Contest/template.cpp"
struct Trie {
    vi2d to;
    vi end, pref;
    int sigma;
    char norm;
    Trie(int sigma_ = 'z' - 'a' + 1, char norm_ = 'a')
        : sigma(sigma_), norm(norm_) {
        to = {vector<int>(sigma)};
        end = {0}, pref = {0};
    }
    int next(int node, char key) { return to[node][key - norm]; }
    void insert(const string &s) {
        int x = 0;
        for (auto c : s) {
            int &nxt = to[x][c - norm];
            if (!nxt) {
                nxt = len(to);
                to.push_back(vi(sigma));
                end.emplace_back(0), pref.emplace_back(0);
            }
            x = nxt, pref[x]++;
        }
        end[x]++, pref[0]++;
    }
    void erase(const string &s) {
        int x = 0;
        for (char c : s) {
            int &nxt = to[x][c - norm];
            x = nxt, pref[x]--;
            if (!pref[x]) nxt = 0;
        }
        end[x]--, pref[0]--;
    }
    int find(const string &s) {
        int x = 0;
        for (auto c : s) {
            x = to[x][c - norm];
            if (!x) return 0;
        }
        return x;
    }
};
```

13.18 Z-function get occurence positions

Time: $O(len(s) + len(p))$

```cpp
#include "../Contest/template.cpp"
vi getOccPos(string& s, string& p) {
    // Z-function
    char delim = '#';
    string t{p + delim + s};
    vi zs(len(t));
    for (int i = 1, l = 0, r = 0; i < len(t); i++) {
        if (i <= r) zs[i] = min(zs[i - l], r - i + 1);
        while (zs[i] + i < len(t) and t[zs[i]] == t[i + zs[i]]) zs[i]++;
        if (r < i + zs[i] - 1) l = i, r = i + zs[i] - 1;
    }
    // Iterate over the results of Z-function to get
    // ranges
    vi ans;
    int start = len(p) + 1 + 1 - 1;
    for (int i = start; i < len(zs); i++) {
        if (zs[i] == len(p)) {
            int l = i - start;
            ans.emplace_back(l);
        }
    }
    return ans;
}
template <class T>
std::vector<int> z_algorithm(const std::vector<T>& s) {
    int n = int(s.size());
    if (n == 0) return {};
    std::vector<int> z(n);
    for (int i = 1, j = 0; i < n; i++) {
        int& k = z[i];
        k = (j + z[j] <= i) ? 0 : std::min(j + z[j] - i, z[i - j]);
        while (i + k < n && s[k] == s[i + k]) k++;
        if (j + z[j] < i + z[i]) j = i;
    }
    z[0] = n;
    return z;
}
std::vector<int> z_algorithm(const std::string& s) {
    int n = int(s.size());
    std::vector<int> s2(n);
    for (int i = 0; i < n; i++) {
        s2[i] = s[i];
    }
    return z_algorithm(s2);
}
// Pattern matching com um erro ??
// prefixo do ãpadro dando certo
// depois o sufixo continua dando certo...
//
//
// Pattern matching com x erros
//
```

```
// únmero de substrings distintas...
//
// únmero de bustrings distintas
// oline ??
//
```

```
// achar o íperodo
//
// kmp é áintercambivel com z
```